LAPORAN PRAKTIKUM

MODUL IX
GRAPH DAN TREE

Disusun oleh:
FATTAH RIZQY ADHIPRATAMA
NIM: 2311102019

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

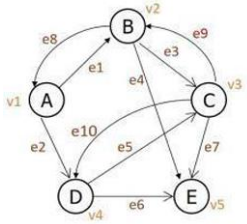
BAB II
DASAR TEORI

1. Graph

Graf adalah suatu struktur data nonlinier yang terdiri dari simpul dan sisi, dimana simpul berisi informasi atau data, dan sisi tersebut berfungsi sebagai penghubung antar pasangan simpul. Ini digunakan untuk memecahkan masalah nyata seperti menemukan rute terbaik ke lokasi tujuan dan rute untuk telekomunikasi dan jejaring sosial. Pengguna dianggap sebagai simpul dalam Grafik, dan kabel adalah sisi yang menghubungkan pengguna.

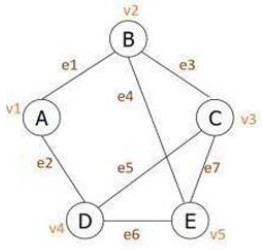
Jenis – Jenis Graf :

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



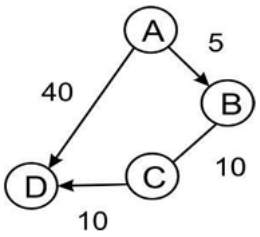
Directed Graph

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



Undirected Graph

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



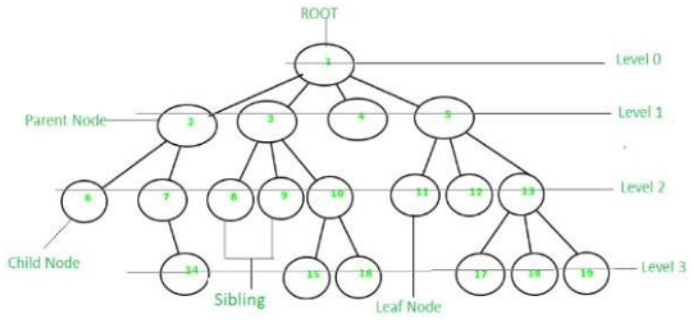
Weight Graph

2. Tree

Tree adalah tipe struktur data yang sifatnya non-linier dan berbentuk hierarki. Mengapa tree disebut sebagai struktur data non-linier? Alasannya karena data pada tree tidak disimpan secara berurutan. Sebaliknya, data diatur pada beberapa level yang disebut struktur hierarkis. Karena itu, tree dianggap sebagai struktur data non-linear.

Hierarki pada struktur tree dapat diibaratkan seperti sebuah pohon keluarga di mana terdapat hubungan antara orang tua dan anak. Titik yang lebih atas disebut simpul induk sedangkan simpul di bawahnya adalah simpul anak. Struktur data tree terdiri atas kumpulan simpul atau node dimana tiap-tiap simpul dari tree digunakan untuk menyimpan nilai dan sebuah list rujukan ke simpul lain yang disebut simpul anak atau child node.

Tiap-tiap simpul dari tree akan dihubungkan oleh sebuah garis hubung yang dalam istilah teknis disebut edge. Biasanya diimplementasikan menggunakan pointer. Simpul pada tree bisa memiliki beberapa simpul anak (child node). Namun, jalan menuju sebuah child node hanya bisa dicapai melalui maksimal 1 node. Apabila sebuah node atau simpul tidak memiliki child node sama sekali maka dinamakan leaf node.



Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

Operasi Pada Tree

1. Create: digunakan untuk membentuk binary tree baru yang masih kosong.
2. Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
3. isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
4. Insert: digunakan untuk memasukkan sebuah node kedalam tree.
5. Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
6. Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

7. Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
8. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
9. Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
10. Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

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 D:\Data Semester 2\Praktikum Struktur Data dan Algoritme\Modul 9>
code.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher
oyw.upo' '--stdout=Microsoft-MIEngine-Out-mnp4wcdj.4id' '--stderr=Mic
=Microsoft-MIEngine-Pid-pxyfeev1.kvh' '--dbgExe=C:\msys64\ucrt64\bin\
 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:\Data Semester 2\Praktikum Struktur Data dan Algoritme\Modul 9>
```

Deskripsi program

Program tersebut adalah implementasi sederhana dari sebuah graf dalam bahasa pemrograman C++. Program ini mendemonstrasikan cara merepresentasikan graf menggunakan array dua dimensi dan fungsi C++. Grafik terarah ini menunjukkan koneksi antar kota di Jawa Barat, Purwokerto, dan Yogyakarta, dengan bobot koneksi mewakili jarak antar kota.

2. Guided 2

Source Code

```
#include <iostream>

using namespace std;

// PROGRAM BINARY TREE
// Deklarasi Pohon
```

```
struct Pohon {
    char data;
    Pohon *left, *right, *parent; //pointer
};

//pointer global
Pohon *root;

// Inisialisasi
void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data) {
    if (isEmpty()) {
        root = newPohon(data);
            cout << "\nNode " << data << " berhasil dibuat
menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
```

```
        if (node->left != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kiri!"<< endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan
ke child kiri " << node->data << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kanan!"<< endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
                cout << "\nNode " << data << " berhasil
ditambahkan ke child kanan " << node->data << endl;
                return baru;
        }
    }
}

void update(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
```

```
            cout << "\nNode yang ingin diganti tidak ada!!" <<
endl;
        else {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah
menjadi " << data << endl;
        }
    }
}

void retrieve(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData 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;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
```

```
    else {
        if (node != NULL) {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat 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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            if (node != root) {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
            cout << "\nNode subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Hapus Tree
void clear() {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat 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) {
    if (isEmpty()) {
        cout << "\nBuat 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 characteristic() {
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

int main() {
    init();
    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 << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;
    cout << "InOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;
    cout << "PostOrder :" << endl;
    postOrder(root);
    cout << "\n" << endl;
    characteristic();
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;
    characteristic();
}
```

Screenshoot Program

```
PS D:\Data Semester 2\Praktikum Struktur Data
Data dan Algoritme\Modul 9\" ; if ($?) { g++ g

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
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:\Data Semester 2\Praktikum Struktur
```

Deskripsi Program

Program tersebut merupakan implementasi dari struktur data Binary Tree menggunakan bahasa C++. Binary Tree adalah struktur data berhirarki di mana setiap node memiliki paling banyak dua anak, yaitu anak kiri dan anak kanan. Memperbarui data dalam node, mengambil data node, dan menemukan informasi lengkap tentang suatu node termasuk parent, sibling, dan child-nya. Selain itu, program menyediakan metode traversal seperti preOrder, inOrder, dan postOrder untuk menjelajahi pohon. Program juga dilengkapi dengan fungsi untuk menghitung ukuran dan tinggi pohon, menghapus subtree, dan menghapus seluruh pohon.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <climits>
#include <algorithm>

using namespace std;

// Struktur untuk merepresentasikan simpul dalam graph
struct Vertex {
    string nama;
    int jarak;
};

// Fungsi untuk membuat graph yang direpresentasikan sebagai
adjacency matrix
vector<vector<int>> createGraph(vector<string>& vertices) {
    // Mendapatkan jumlah simpul dari user
    int numVertices2311102019;
    cout << "Masukkan jumlah simpul: ";
    cin >> numVertices2311102019;

    // Memasukkan nama simpul
    vertices.resize(numVertices2311102019);
    for (int i = 0; i < numVertices2311102019; i++) {
        cout << "Masukkan nama simpul " << i + 1 << ": ";
        cin >> vertices[i];
    }

    // Membangun adjacency matrix
    vector<vector<int>> adjacencyMatrix(numVertices2311102019,
vector<int>(numVertices2311102019));
    for (int i = 0; i < numVertices2311102019; i++) {
        for (int j = 0; j < numVertices2311102019; j++) {
            if (i == j) {
```

```
                // Bobot untuk simpul yang sama selalu 0
                adjacencyMatrix[i][j] = 0;
            } else {
                cout << "Masukkan bobot " << vertices[i] << "-->"
<< vertices[j] << ": ";
                cin >> adjacencyMatrix[i][j];
            }
        }
    }

    return adjacencyMatrix;
}

// Fungsi untuk mencari jarak antara dua kota (simpul) dalam
graph
int findDistance(const vector<vector<int>>& adjacencyMatrix,
const vector<string>& vertices,
                 const string& startCity, const string& endCity)
{
    // Mencari indeks kota awal dan kota tujuan dalam daftar
simpul
    auto startIt = find(vertices.begin(), vertices.end(),
startCity);
    auto endIt = find(vertices.begin(), vertices.end(), endCity);

    if (startIt == vertices.end() || endIt == vertices.end()) {
        return -1; // Kota tidak ditemukan
    }

    int startIndex = distance(vertices.begin(), startIt);
    int endIndex = distance(vertices.begin(), endIt);

    // Menghitung jarak menggunakan algoritma Dijkstra
    vector<int> distances(vertices.size(), INT_MAX);
    distances[startIndex] = 0;

    // Menginisialisasi set simpul yang belum dikunjungi
    set<int> unvisitedVertices;
    for (int i = 0; i < vertices.size(); i++) {
        unvisitedVertices.insert(i);
    }
```

```
    // Menjalankan algoritma Dijkstra
    while (!unvisitedVertices.empty()) {
        // Menemukan simpul dengan jarak terpendek dari simpul
awal yang belum dikunjungi
        int currentVertex =
*min_element(unvisitedVertices.begin(), unvisitedVertices.end(),
                                          [&](int i, int j) {
return distances[i] < distances[j]; });

        // Menghapus simpul dari set simpul yang belum dikunjungi
        unvisitedVertices.erase(currentVertex);

        // Memperbarui jarak simpul yang terhubung dengan simpul
saat ini
        for (int neighbor = 0; neighbor < vertices.size();
neighbor++) {
            if (adjacencyMatrix[currentVertex][neighbor] > 0 &&
                distances[currentVertex] +
adjacencyMatrix[currentVertex][neighbor] < distances[neighbor]) {
                distances[neighbor] = distances[currentVertex] +
adjacencyMatrix[currentVertex][neighbor];
            }
        }
    }

    // Mengembalikan jarak ke kota tujuan
    return distances[endIndex];
}

int main() {
    // Membangun graph
    vector<string> vertices;
    vector<vector<int>> adjacencyMatrix = createGraph(vertices);

    // Mencari jarak antara dua kota
    string kotaAwal, kotaAkhir;
    cout << "Masukkan nama kota awal: ";
    cin >> kotaAwal;
    cout << "Masukkan nama kota tujuan: ";
    cin >> kotaAkhir;
```

```
    int distance = findDistance(adjacencyMatrix, vertices,
kotaAwal, kotaAkhir);

    // Menampilkan hasil
    if (distance == -1) {
        cout << "Tidak ada jalur dari " << kotaAwal << " ke " <<
kotaAkhir << "." << endl;
    } else {
        cout << "Jarak dari " << kotaAwal << " ke " << kotaAkhir
<< " adalah " << distance << " kilometer." << endl;
    }

    return 0;
}
```

Screenshoot program

```
PS D:\Data Semester 2\Praktikum Struktur Data dan Algor
me\Modul 9\" ; if ($?) { g++ unguided1.cpp -o unguided1
Masukkan jumlah simpul: 2
Masukkan nama simpul 1: PURWOKERTO
Masukkan nama simpul 2: SEMARANG
Masukkan bobot PURWOKERTO-->SEMARANG: 10
Masukkan bobot SEMARANG-->PURWOKERTO: 8
Masukkan nama kota awal: PURWOKERTO
Masukkan nama kota tujuan: SEMARANG
Jarak dari PURWOKERTO ke SEMARANG adalah 10 kilometer.
```

Deskrirpsi Program

Program ini adalah implementasi dari graf untuk menghubungkan antar kota (simpul) dengan bobot (jarak) antar kota yang di inputkan oleh pengguna. Program ini memungkinkan pengguna untuk memasukkan nama-nama kota dan jarak antara kota-kota tersebut, dan kemudian mencari jarak terpendek antara dua kota yang dimasukkan oleh pengguna. Program ini membantu dalam perencanaan rute atau perjalanan antar kota dalam sebuah wilayah.

2. Unguided 2

Source Code

```
#include <iostream>
```

```cpp
#include <queue>
#include <vector>

using namespace std;

// Deklarasi Pohon
struct Pohon {
    char data_2311102019;
    Pohon *left, *right, *parent; // pointer
};

// pointer global
Pohon *root;

// Inisialisasi
void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

Pohon *newPohon(char data_2311102019) {
    Pohon *node = new Pohon();
    node->data_2311102019 = data_2311102019;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data_2311102019) {
    if (isEmpty()) {
        root = newPohon(data_2311102019);
        cout << "\nNode " << data_2311102019 << " berhasil
dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}
```

```
Pohon *insertLeft(char data_2311102019, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->left != NULL) {
            cout << "\nNode " << node->data_2311102019 << "
sudah ada child kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data_2311102019);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data_2311102019 << " berhasil
ditambahkan ke child kiri " << node->data_2311102019 << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data_2311102019, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data_2311102019 << "
sudah ada child kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data_2311102019);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data_2311102019 << " berhasil
ditambahkan ke child kanan " << node->data_2311102019 << endl;
            return baru;
        }
    }
}
```

```
void update(char data_2311102019, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" <<
endl;
        else {
            char temp = node->data_2311102019;
            node->data_2311102019 = data_2311102019;
            cout << "\nNode " << temp << " berhasil diubah
menjadi " << data_2311102019 << endl;
        }
    }
}

void retrieve(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData node : " << node->data_2311102019
<< endl;
        }
    }
}

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data_2311102019
<< endl;
            cout << "Root : " << root->data_2311102019 << endl;
            if (!node->parent)
```

```
                cout << "Parent : (tidak punya parent)" <<
endl;
            else
                cout << "Parent : " << node->parent-
>data_2311102019 << endl;
            if (node->parent != NULL && node->parent->left !=
node && node->parent->right == node)
                cout << "Sibling : " << node->parent->left-
>data_2311102019 << endl;
            else if (node->parent != NULL && node->parent-
>right != node && node->parent->left == node)
                cout << "Sibling : " << node->parent->right-
>data_2311102019 << 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_2311102019 << endl;
            if (!node->right)
                cout << "Child Kanan : (tidak punya Child
kanan)" << endl;
            else
                cout << "Child Kanan : " << node->right-
>data_2311102019 << endl;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            cout << " " << node->data_2311102019 << ", ";
            preOrder(node->left);
```

```
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            inOrder(node->left);
            cout << " " << node->data_2311102019 << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data_2311102019 << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            if (node != root) {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
```

```
                    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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data_2311102019 << "
berhasil dihapus." << endl;
    }
}

// Hapus Tree
void clear() {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat 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) {
    if (isEmpty()) {
        cout << "\nBuat 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 characteristic() {
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
```

```
}

// Menampilkan Child dari node
void showChild(Pohon *node) {
    if (node->left != NULL)
        cout << "Child Kiri : " << node->left->data_2311102019
<< endl;
    else
        cout << "Child Kiri : (tidak punya Child kiri)" <<
endl;

    if (node->right != NULL)
        cout << "Child Kanan : " << node->right-
>data_2311102019 << endl;
    else
        cout << "Child Kanan : (tidak punya Child kanan)" <<
endl;
}

// Menampilkan Descendant dari node
void showDescendants(Pohon *node) {
    if (node == NULL)
        return;

    queue<Pohon*> q;
    q.push(node);
    vector<char> descendants;

    while (!q.empty()) {
        Pohon* current = q.front();
        q.pop();

        if (current != node) {
            descendants.push_back(current->data_2311102019);
        }

        if (current->left != NULL) {
            q.push(current->left);
        }

        if (current->right != NULL) {
```

```
            q.push(current->right);
        }
    }

    if (descendants.empty()) {
        cout << "Node " << node->data_2311102019 << " tidak
punya descendants." << endl;
    } else {
        cout << "Descendants of node " << node->data_2311102019
<< " : ";
        for (char data_2311102019 : descendants) {
            cout << data_2311102019 << " ";
        }
        cout << endl;
    }
}

// Fungsi untuk menampilkan menu dan mengelola input pengguna
void menu() {
    char data_2311102019;
    Pohon *node;
    int pilihan;

    do {
        cout << "\n--- Menu ---" << endl;
        cout << "1. Buat Node Root" << endl;
        cout << "2. Insert Left" << endl;
        cout << "3. Insert Right" << endl;
        cout << "4. Update Node" << endl;
        cout << "5. Retrieve Node" << endl;
        cout << "6. Find Node" << endl;
        cout << "7. PreOrder Traversal" << endl;
        cout << "8. InOrder Traversal" << endl;
        cout << "9. PostOrder Traversal" << endl;
        cout << "10. Tampilkan Child Node" << endl;
        cout << "11. Tampilkan Descendants Node" << endl;
        cout << "12. Hapus SubTree" << endl;
        cout << "13. Hapus Tree" << endl;
        cout << "14. Karakteristik Tree" << endl;
        cout << "15. Keluar" << endl;
        cout << "Pilih opsi: ";
```

```
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan data untuk root: ";
                cin >> data_2311102019;
                buatNode(data_2311102019);
                break;
            case 2:
                cout << "Masukkan data untuk child kiri: ";
                cin >> data_2311102019;
                cout << "Masukkan data parent: ";
                cin >> data_2311102019;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    insertLeft(data_2311102019, node);
                } else {
                    cout << "Node parent tidak ditemukan!" <<
endl;
                }
                break;
            case 3:
                cout << "Masukkan data untuk child kanan: ";
                cin >> data_2311102019;
                cout << "Masukkan data parent: ";
                cin >> data_2311102019;
                node = root;
```

```
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    insertRight(data_2311102019, node);
                } else {
                    cout << "Node parent tidak ditemukan!" <<
endl;
                }
                break;
            case 4:
                cout << "Masukkan data node yang akan diupdate:
";
                cin >> data_2311102019;
                cout << "Masukkan data baru: ";
                char newData;
                cin >> newData;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
```

```
                if (node && node->data_2311102019 ==
data_2311102019) {
                    update(newData, node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            case 5:
                cout << "Masukkan data node yang akan
diretrieve: ";
                cin >> data_2311102019;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    retrieve(node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            case 6:
                cout << "Masukkan data node yang akan dicari:
";
                cin >> data_2311102019;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
```

```
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    find(node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            case 7:
                cout << "\nPreOrder :" << endl;
                preOrder(root);
                cout << "\n" << endl;
                break;
            case 8:
                cout << "InOrder :" << endl;
                inOrder(root);
                cout << "\n" << endl;
                break;
            case 9:
                cout << "PostOrder :" << endl;
                postOrder(root);
                cout << "\n" << endl;
                break;
            case 10:
                cout << "Masukkan data node: ";
                cin >> data_2311102019;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
```

```
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    showChild(node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            case 11:
                cout << "Masukkan data node: ";
                cin >> data_2311102019;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    showDescendants(node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            case 12:
                cout << "Masukkan data node untuk menghapus
subtree: ";
                cin >> data_2311102019;
                node = root;
                while (node && node->data_2311102019 !=
data_2311102019) {
```

```
                    if (node->left && node->left-
>data_2311102019 == data_2311102019) {
                        node = node->left;
                    } else if (node->right && node->right-
>data_2311102019 == data_2311102019) {
                        node = node->right;
                    } else {
                        break;
                    }
                }
                if (node && node->data_2311102019 ==
data_2311102019) {
                    deleteSub(node);
                } else {
                    cout << "Node tidak ditemukan!" << endl;
                }
                break;
            case 13:
                clear();
                break;
            case 14:
                characteristic();
                break;
            case 15:
                cout << "Keluar..." << endl;
                break;
            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 15);
}

int main() {
    init();
    menu();
    return 0;
}
```

Screenshoot Program

```
PS D:\Data Semester 2\Praktikum Struktur Data dan Algoritme\Modul 9>
code.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher
w1x.u2m' '--stdout=Microsoft-MIEngine-Out-5zdnvy5x.vi3' '--stderr=Mic
=Microsoft-MIEngine-Pid-q1zsj1lz.e3q' '--dbgExe=C:\msys64\ucrt64\bin\

--- Menu ---
1. Buat Node Root
2. Insert Left
3. Insert Right
4. Update Node
5. Retrieve Node
6. Find Node
7. PreOrder Traversal
8. InOrder Traversal
9. PostOrder Traversal
10. Tampilkan Child Node
11. Tampilkan Descendants Node
12. Hapus SubTree
13. Hapus Tree
14. Karakteristik Tree
15. Keluar
Pilih opsi: 10
Masukkan data node: f
Node tidak ditemukan!
```

Deskripsi Program

Program ini adalah implementasi dari struktur data tree menggunakan C++. Program ini memungkinkan pengguna untuk membuat, mengelola, dan melakukan operasi pada pohon biner, termasuk membuat node, menyisipkan node ke kiri atau kanan, memperbarui nilai node, mengambil nilai node, mencari node, melakukan penelusuran (traversal) preOrder, inOrder, dan postOrder, menampilkan child dari suatu node, menampilkan descendants dari suatu node, menghapus subtree, menghapus seluruh pohon, dan menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node.

BAB IV

KESIMPULAN

Graf dan tree merupakan struktur data fundamental yang memiliki peran penting dalam berbagai aplikasi. Pemahaman yang baik tentang struktur, operasi, dan algoritma yang terkait dengan graf dan tree sangatlah penting bagi para programmer dan ilmuwan komputer. Praktikum graf dan tree memberikan kesempatan untuk mempelajari konsep-konsep tersebut secara mendalam dan menerapkannya dalam contoh-contoh konkret.

Baik graf maupun tree memiliki banyak aplikasi dalam dunia nyata. Contohnya, graf digunakan dalam pemetaan jalan, jejaring sosial, perencanaan rute, dan optimasi jaringan. Sedangkan tree digunakan dalam pemrosesan bahasa alami, database, komputasi grafis, dan pemodelan hierarki data. Keduanya dapat digunakan untuk merepresentasikan hubungan antara entitas dalam sebuah sistem, seperti jaringan komputer, relasi antar objek, hierarki data, dan sebagainya.

DAFTAR PUSTAKA

[1] Alyssa, 2024. Struktur Data Grafik dan Algorithms (Contoh). Diakses pada tanggal 8 Juni 2024, dari https://www.guru99.com/id/graphs-in-data-structures.html

[2] Trivusi, 2022. Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya. Diakses pada tanggal 8 Juni 2024, dari https://www.trivusi.web.id/2022/07/struktur-data-tree.html