LAPORAN PRAKTIKUM
MODUL 9
GRAPH DAN TREE

Disusun oleh :

Diva Octaviani

NIM : 2311102006

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

a. Pengertian Graph

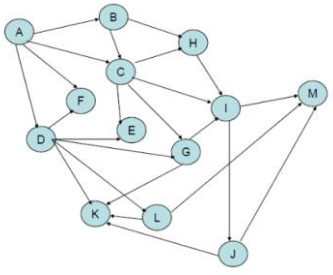
Teori graph atau teori grafik dalam matematika dan ilmu komputer adalah cabang kajian yang mempelajari sifat-sifat "graf" atau "grafik". Ini tidak sama dengan "Grafika". Secara informal, suatu graf adalah himpunan benda-benda yang disebut "simpul" (vertex atau node) yang terhubung oleh "sisi" (edge) atau "busur" (arc). Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan "simpul") yang dihubungkan oleh garis-garis (melambangkan "sisi") atau garis berpanah (melambangkan "busur"). Suatu sisi dapat menghubungkan suatu simpul dengan simpul yang sama. Sisi yang demikian dinamakan "gelang" (loop).

b. Jenis-jenis Graph

- **Graph Berarah (Directed Graph)**

  Disebut graph berarah jika sisi-sisi graph hanya berlaku satu arah. Misalnya : {x,y} yaitu arah x ke y, bukan dari y ke x; x disebut origin dan y disebut terminus. Secara notasi sisi digraph ditulis sebagai vektor (x, y).

  Contoh Digraph G = {V, E} : V = {A, B, C, D, E, F, G, H, I,J, K, L, M} E = {(A,B),(A,C), (A,D), (A,F), (B,C), (B,H), (C,E), (C,G), (C,H), (C,I), (D,E), (D,F), (D,G), (D,K), (D,L), 90 (E,F), (G,I), (G,K), (H,I), (I,J), (I,M), (J,K), (J,M), (L,K), (L,M)}.

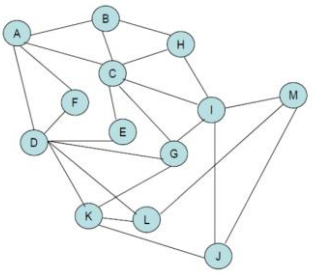


Gambar 1. Graph berarah

- **Graph Tak Berarah (Undirected Graph)**

  Setiap sisi {x, y} berlaku pada kedua arah, baik x ke y maupun y ke x. Secara grafis, sisi pada undigraph tidak memiliki mata panah dan secara notasional menggunakan kurung kurawal.

  Contoh Undigraph G = {V, E} V = {A, B, C, D, E, F, G, H, I,J, K, L, M} E = { {A,B},{A,C}, {A,D}, {A,F}, {B,C}, {B,H}, {C,E}, {C,G}, {C,H}, {C,I}, {D,E}, {D,F}, {D,G}, {D,K}, {D,L}, {E,F}, {G,I}, {G,K}, {H,I}, {I,J}, {I,M}, {J,K}, {J,M}, {L,K}, {L,M}}.



Gambar 2. Graph tak berarah

- **Graph Berbobot (Weighted Graph)**

  Graph dengan sisi mempunyai Bobot/ Biaya. “Biaya" ini bisa mewakili banyak aspek: biaya ekonomi suatu aktifitas, jarak geografis/tempuh, waktu tempuh, tingkat kesulitan, dan lain sebagainya.

  Contoh : Graph ini merupakan Undirected Weighted Graph. Order dari verteks A = 4, verteks B = 3, dst. Adjacentcy list dari D adalah = {A, E, F, G, K, L}.

2. Tree

a. Definisi Tree

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hierarkis (hubungan one to many) antara elemen-elemen. Tree bias didefinisikan sebagai kumpulan simpul/node dengan elemen khusus yang disebut Root. Node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lain (disebut Subtree). Untuk lebih jelasnya, di bawah ini akan diuraikan istilah-istilah umum dalam tree.

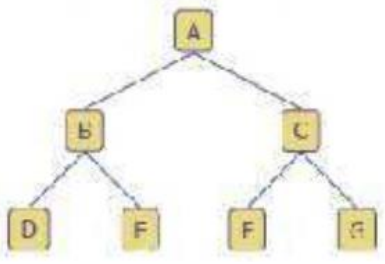
Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada dibawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama dengan suatu node
Subtree	Bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan / level dalam suatu tree
Root	Satu-satunya node khusus dalam tree yang tak punyak predecessor
Leaf	Node-node dalam tree yang tak memiliki successor
Degree	Banyaknya child yang dimiliki suatu node

Gambar 3. Istilah umum dalam pohon

b. Jenis-jenis Tree

- **Full Binary Tree**

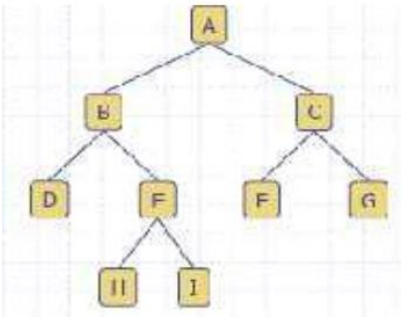
  Jenis binary tree ini tiap nodenya (kecuali leaf) memiliki dua child dan tiap subtree harus mempunyai panjang path yang sama.



Gambar 4. Full binary tree

- **Complete Binary Tree**

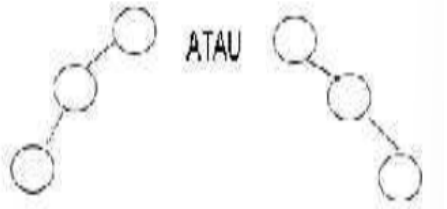
  Jenis ini mirip dengan Full Binary Tree, namun tiap subtree boleh memiliki panjang path yang berbeda dan setiap node kecuali leaf hanya boleh memiliki 2 child.



Gambar 5. Complete binary tree

- **Skewed Binary Tree**

  Skewed Binary Tree adalah Binary Tree yang semua nodenya (kecuali leaf) hanya memiliki satu child.



Gambar 6. Skewed binary tree

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 D:\Praktikum Struktur Data\Modul 9> cd "d:\Praktikum Struktur
Data\Modul 9\" ; if ($?) { g++ guided1.cpp -o guided1 } ; if ($?)
 { .\guided1 }
 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)
```

Deskripsi program

Program tersebut merupakan sebuah implementasi sederhana dari struktur data graph. Graph yang digunakan dalam kode program ini terdiri dari 7 simpul yang direpresentasikan dalam array simpul dan relasi antar simpul yang direpresentasikan dalam matriks busur. Fungsi tampilGraph() digunakan untuk menampilkan graph ke layar. Fungsi ini menggunakan perulangan for untuk mengakses setiap elemen dalam matriks busur. Jika nilai elemen tersebut tidak sama dengan 0, maka simpul yang terhubung dengan elemen tersebut akan ditampilkan beserta bobot relasinya.

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
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
    {
        // 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

```
PS D:\Praktikum Struktur Data\Modul 9> cd "d:\Praktikum Struktur
Data\Modul 9\" ; if ($?) { g++ guided2.cpp -o guided2 } ; if ($?)
 { .\guided2 }

 Node A berhasil dibuat menjadi root.

 Node B berhasil ditambahkan ke child kiri A

 Node C berhasil ditambahkan ke child kanan A

 Node D berhasil ditambahkan ke child kiri B

 Node E berhasil ditambahkan ke child kanan B

 Node F berhasil ditambahkan ke child kiri C

 Node G berhasil ditambahkan ke child kiri E

 Node H berhasil ditambahkan ke child kanan E

 Node I berhasil ditambahkan ke child kiri G

 Node J berhasil ditambahkan ke child kanan G

 Node C berhasil diubah menjadi Z

 Node Z berhasil diubah menjadi C
```

```
Data node : C

Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)
```

```
PreOrder :
A,  B,  D,  E,  G,  I,  J,  H,  C,  F,

InOrder :
D,  B,  I,  G,  J,  E,  H,  A,  F,  C,

PostOrder :
D,  I,  J,  G,  H,  E,  B,  F,  C,  A,


Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.

PreOrder :
A,  B,  D,  E,  C,  F,


Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
```

Deskripsi program

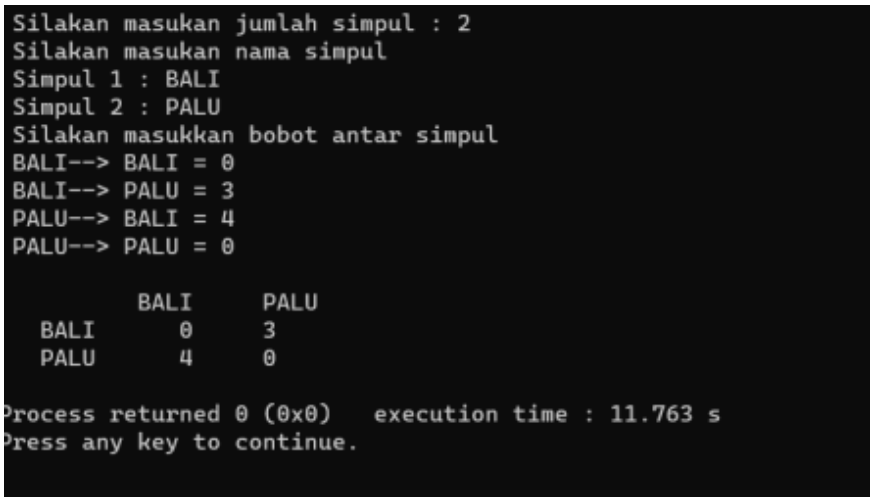
Program tersebut mengimplementasikan Binary Tree dimulai dengan mendefinisikan struktur data Pohon yang terdiri dari node dengan data karakter, dan pointer ke node kiri, kanan, dan parent. Program menyediakan berbagai fungsi untuk menginisialisasi, membuat node baru, menambahkan node kiri dan kanan, mengupdate data node, mengambil dan mencari data node, melakukan penelurusan preOrder, inOrder, dan postOrder, menghapus subtree dan keseluruhan tree, serta memeriksa ukuran dan tinggi tree.

LATIHAN KELAS – UNGUIDED

1. Unguided 1

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

Output program



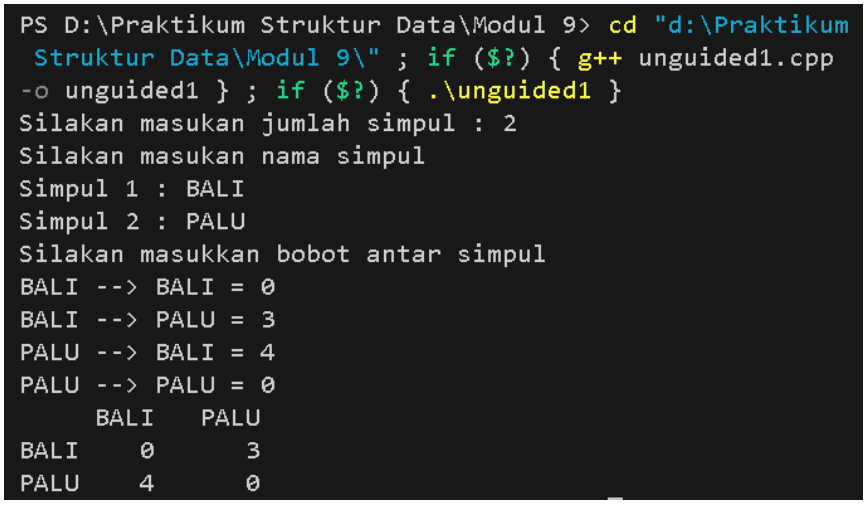
Source code

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main() {
    int jumlah_simpul;

    cout << "Silakan masukan jumlah simpul : ";
    cin >> jumlah_simpul;
    string nama_simpul[jumlah_simpul];

    cout << "Silakan masukan nama simpul\n";

    cin.ignore();
```

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

    int simpul[jumlah_simpul][jumlah_simpul];

    cout << "Silakan masukkan bobot antar simpul\n";

    for (int i = 0; i < jumlah_simpul; i++) {
        for (int k = 0; k < jumlah_simpul; k++) {
            cout <<  nama_simpul[i] << " --> " <<
nama_simpul[k] << " = ";
            cin >> simpul[i][k];
        }
    }

    // CHECKS LONGEST LINE IN THE ARRAY OF CITIES
    // FOR SETW()
    int formatting = nama_simpul[0].length()+1;

    for (int i = 0; i < jumlah_simpul-1; i++) {
        if (nama_simpul[i].length() <
nama_simpul[i+1].length())  {
            formatting = nama_simpul[i+1].length()+1;
        }
    }

    // JUDUL TABEL
    cout << left << setw(formatting) << " ";
    for (int i = 0; i < jumlah_simpul; i++) {
        cout << left << setw(nama_simpul[i].length()) <<
nama_simpul[i]<< "   ";
    }
```

```
    cout << endl;

    // TOTAL KOTA FOR FORMATTING
    int DivaOctaviani_2311102006 = sizeof(nama_simpul) /
sizeof(*nama_simpul);

    // PRINT ISI TABEL
    for (int i = 0; i < jumlah_simpul; i++) {
        cout << left << setw(formatting) << nama_simpul[i];
        for (int k = 0; k < DivaOctaviani_2311102006; k++) {
            cout << right << setw(nama_simpul[k].length()) <<
simpul[i][k] << "   ";
        }
        cout << endl;
    }
}
```

Screenshoot program



Deskripsi program

Program tersebut menggunakan konsep graph untuk merepresentasikan hubungan antara simpul-simpul (nodes) dan bobot (weight) antara setiap pasangan simpul.

Secara lebih spesifik, program menggunakan array dua dimensi simpul[jumlah_simpul][jumlah_simpul] untuk menyimpan bobot antara setiap pasangan simpul. Setiap elemen simpul[i][j] menyimpan bobot dari simpul ke-i menuju simpul ke-j.

.

2. Unguided 2

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

Source code

```
#include <iostream>
#include <vector>
using namespace std;

// Declaring the Tree structure
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root = nullptr;

// Initialize the tree
void init()
{
    root = NULL;
}

// Check if the tree is empty
int isEmpty()
{
    return (root == NULL) ? 1 : 0;
}

// Create a new node
Pohon *buatNode(char data)
{
    Pohon *newNode = new Pohon();
    newNode->data = data;
```

```
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->parent = NULL;
    // cout << "\nNode " << data << " berhasil dibuat." <<
endl;
    return newNode;
}

// Insert a node to the left
Pohon *insertLeft(Pohon *parent, Pohon *child)
{
    if (isEmpty() == 1)
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (parent->left != NULL)
        {
            cout << "\nNode " << parent->left->data << " sudah
ada child kiri!" << endl;
            return NULL;
        }
        else
        {
            child->parent = parent;
            parent->left = child;
            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kiri " << child->parent->data << endl;
            return child;
        }
    }
}
```

```
// Insert a node to the right
Pohon *insertRight(Pohon *parent, Pohon *child)
{
    if (root == NULL)
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (parent->right != NULL)
        {
            cout << "\nNode " << parent->right->data << " sudah
ada child kanan!" << endl;
            return NULL;
        }
        else
        {
            child->parent = parent;
            parent->right = child;
            // cout << "\nNode " << child->data << " berhasil
ditambahkan ke child kanan " << child->parent->data << endl;
            return child;
        }
    }
}

// Update node data
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        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;
        }
    }
}

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

```
// Find node and display its properties
void find(Pohon *node)
{
    if (!root)
    {
        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;
        }
    }
}

// Pre-order traversal
void preOrder(Pohon *node)
{
    if (node != NULL)
    {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

// In-order traversal
void inOrder(Pohon *node)
{
    if (node != NULL)
    {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
```

```
    }
}

// Post-order traversal
void postOrder(Pohon *node)
{
    if (node != NULL)
    {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Delete the entire tree
void deleteTree(Pohon *node)
{
    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;
```

```
        }
    }
}

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

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

// Get the size of the tree
int size(Pohon *node)
{
    if (node == NULL)
```

```
    {
        return 0;
    }
    else
    {
        return 1 + size(node->left) + size(node->right);
    }
}

// Get the height of the tree
int height(Pohon *node)
{
    if (node == NULL)
    {
        return 0;
    }
    else
    {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return (heightKiri >= heightKanan) ? heightKiri + 1 :
heightKanan + 1;
    }
}

// Display tree characteristics
void charateristic()
{
    cout << "\nSize Tree : " << size(root) << endl;
    cout << "Height Tree : " << height(root) << endl;
    cout << "Average Node of Tree : " << (size(root) /
(float)height(root)) << endl;
}
```

```
int main()
{
    root = buatNode('A');
    int menu, part, part2;
    char DivaOctaviani_2311102006;

    vector<Pohon *> nodes;
    nodes.push_back(buatNode('B'));
    nodes.push_back(buatNode('C'));
    nodes.push_back(buatNode('D'));
    nodes.push_back(buatNode('E'));
    nodes.push_back(buatNode('F'));
    nodes.push_back(buatNode('G'));
    nodes.push_back(buatNode('H'));
    nodes.push_back(buatNode('I'));
    nodes.push_back(buatNode('J'));

    insertLeft(root, nodes[0]);
    insertRight(root, nodes[1]);
    insertLeft(nodes[0], nodes[2]);
    insertRight(nodes[0], nodes[3]);
    insertLeft(nodes[1], nodes[4]);
    insertLeft(nodes[3], nodes[5]);
    insertRight(nodes[3], nodes[6]);
    insertLeft(nodes[5], nodes[7]);
    insertRight(nodes[5], nodes[8]);

    // update('Z', nodes[1]);
    // update('C', nodes[1]);

    do
    {
        cout << "\n---------- PROGHRAM TREE ----------\n"
                "1. Tambah node\n"
```

```
                "2. Tambah di kiri\n"
                "3. Tambah di kanan\n"
                "4. Lihat karakteristik tree\n"
                "5. Lihat isi data tree\n"
                "6. Cari data tree\n"
                "7. Penelurusan (Traversal) preOrder\n"
                "8. Penelurusan (Traversal) inOrder\n"
                "9. Penelurusan (Traversal) postOrder\n"
                "10. Hapus subTree\n"
                "0. KELUAR\n"
                "\nPilih : ";
        cin >> menu;
        cout << "--------Running Command.........\n";
        switch (menu)
        {
        case 1:
            cout << "\n Nama Node (Character) : ";
            cin >> DivaOctaviani_2311102006;

nodes.push_back(buatNode(DivaOctaviani_2311102006));
            break;
        case 2:
            cout << "\nMasukkan nomor untuk node parent : ";
            cin >> part;
            cout << "\nMasukkan nomor untuk node child : ";
            cin >> part2;
            insertLeft(nodes[part], nodes[part2]);
            break;
        case 3:
            cout << "\nMasukkan nomor untuk node parent : ";
            cin >> part;
            cout << "\nMasukkan nomor untuk node child : ";
            cin >> part2;
            insertRight(nodes[part], nodes[part2]);
```

```
            break;
        case 4:
            charateristic();
            break;
        case 5:
            cout << "\nMasukkan nomor node : ";
            cin >> part;
            retrieve(nodes[part]);
            break;
        case 6:
            cout << "\nMasukkan nomor node : ";
            cin >> part;
            find(nodes[part]);
            break;
        case 7:
            cout << "\nPreOrder :" << endl;
            preOrder(root);
            cout << "\n"
                 << endl;
            break;
        case 8:
            cout << "\nInOrder :" << endl;
            inOrder(root);
            cout << "\n"
                 << endl;
            break;
        case 9:
            cout << "\nPostOrder :" << endl;
            postOrder(root);
            cout << "\n"
                 << endl;
            break;
        case 10:
            cout << "\nMasukkan nomor node : ";
```

```
            cin >> part;
            deleteSub(nodes[part]);
            break;
        default:
            break;
        }
    } while (menu != 0);

    // retrieve(nodes[1]);
    // find(nodes[1]);

    // cout << "\nPreOrder :" << endl;
    // preOrder(root);
    // cout << "\n" << endl;

    // cout << "InOrder :" << endl;
    // inOrder(root);
    // cout << "\n" << endl;

    // cout << "PostOrder :" << endl;
    // postOrder(root);
    // cout << "\n" << endl;

    // charateristic();

    // deleteSub(nodes[3]);

    // cout << "\nPreOrder :" << endl;
    // preOrder(root);
    // cout << "\n" << endl;

    // charateristic();
}
```

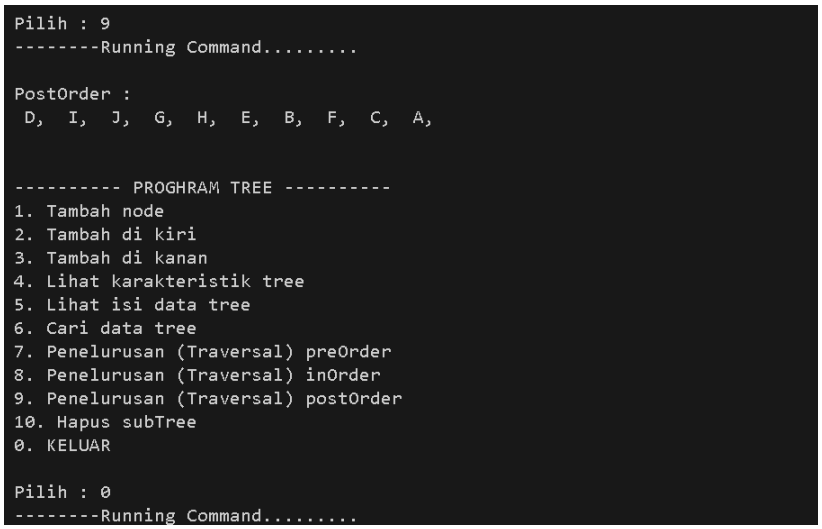
Screenshoot program

```
PS D:\Praktikum Struktur Data\Modul 9> cd "d:\Praktikum Struktur Data\Modul
 9\" ; if ($?) { g++ unguided2.cpp -o unguided2 } ; if ($?) { .\unguided2 }

---------- PROGHRAM TREE ----------
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 4
--------Running Command.........

Size Tree : 10
Height Tree : 5
Average Node of Tree : 2
```

```
---------- PROGHRAM TREE ----------
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 5
--------Running Command.........

Masukkan nomor node : 1

Data node : C

---------- PROGHRAM TREE ----------
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR
```

```
Pilih : 6
--------Running Command.........

Masukkan nomor node : 7

Data Node : I
Root : A
Parent : G
Sibling : J
Child Kiri : (tidak punya Child kiri)
Child Kanan : (tidak punya Child kanan)

---------- PROGHRAM TREE ----------
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 7
--------Running Command.........

PreOrder :
 A,  B,  D,  E,  G,  I,  J,  H,  C,  F,
```

```
---------- PROGHRAM TREE ----------
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 8
--------Running Command.........

InOrder :
 D,  B,  I,  G,  J,  E,  H,  A,  F,  C,


---------- PROGHRAM TREE ----------
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR
```



Deskripsi program

Program tersebut telah mengimplementasikan struktur data pohon biner (binary tree). Setiap node dalam pohon biner direpresentasikan menggunakan struktur Pohon yang memiliki atribut data untuk menyimpan nilai node, serta pointer left, right, dan parent untuk menyimpan referensi ke anak kiri, anak kanan, dan induk dari node tersebut. Program menyediakan beragam fungsi untuk mengelola pohon biner, seperti membuat node baru, menambahkan node sebagai anak kiri atau kanan, memperbarui data node, mendapatkan data node, mencari node, melakukan traversal (pre-order, in-order, dan post-order), menghapus subtree, menghapus keseluruhan pohon, mendapatkan ukuran pohon, dan mendapatkan tinggi pohon.

BAB IV
KESIMPULAN

Graph dan tree merupakan dua struktur data yang penting dalam ilmu komputer dan pemrograman. Keduanya digunakan untuk merepresentasikan hubungan antara objek-objek atau data-data. Namun, terdapat perbedaan mendasar antara keduanya. Graph merupakan struktur data yang lebih umum dan fleksibel, di mana simpul-simpulnya dapat terhubung secara bebas dan dapat membentuk siklus atau loop. Sementara itu, tree memiliki struktur yang lebih ketat dan hierarkis, di mana setiap node (kecuali root) hanya memiliki satu induk dan tidak ada siklus yang terbentuk. Perbedaan lainnya terletak pada cara pengaksesan datanya. Pada graph, pengaksesan dilakukan dengan menelusuri busur (edge), sedangkan pada tree, pengaksesan dilakukan dengan bergerak dari node ke node secara hierarkis, melalui induk atau anak-anaknya. Pemilihan struktur data yang tepat antara graph atau tree tergantung pada sifat dan kebutuhan permasalahan yang ingin diselesaikan.

DAFTAR PUSTAKA

Asisten Praktikum, "*Modul 9 Graph dan Tree*", Learning Management System, 2024.

Sindar, Anita. (2018). *STRUKTUR DATA DAN ALGORITMA DENGAN C++*. Banten: CV. AA. RIZKY.

Muliono, Rizki. (2018). *PEMROGRAMAN C++: ALGORITMA & STRUKTUR DATA*. Universitas Medan Area: Teknik Informatika.

Latifah, Fitri. (2016). *PENERAPAN ALGORITHMA POHON UNTUK OPERASI PENGOLAHANDAN PENYIMPANAN DATA DALAM TEKNIK PEMROGRAMAN (kajian algorithma pohon pada teknik pemrograman)*. Jurnal Techno Nusa Mandiri, 13(2), 23-32.