

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



Disusun Oleh :

NAMA : GALIH TRISNA
NIM : 2311102050

Dosen

Wahyu Andi Saputra, S.Pd., M.Eng

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFROMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

A. Dasar Teori

1. Graf

Graf dalam pemrograman adalah struktur data yang merepresentasikan hubungan antara berbagai entitas atau objek. Sebuah graf terdiri dari simpul (nodes atau vertices) dan sisi (edges) yang menghubungkan pasangan simpul. Graf digunakan dalam berbagai aplikasi ilmu komputer seperti pencarian jalur terpendek, jaringan sosial, dan optimasi.

Terdapat beberapa jenis graf:

- Graf Berarah: Graf di mana setiap tepi memiliki arah, menunjukkan aliran atau orientasi.
- Graf Tak Berarah: Graf di mana tepi tidak memiliki arah, sehingga hubungan antara simpul bersifat dua arah.
- Graf Berbobot: Graf di mana setiap tepinya memiliki nilai atau bobot, yang bisa merepresentasikan jarak, biaya, atau atribut lainnya.

Untuk mengimplementasikan graf dalam kode, ada dua metode umum yang digunakan yaitu menggunakan matriks ketetanggaan (adjacency matrix) atau daftar ketetanggaan (adjacency list).

2. Pohon (Tree)

Pohon (tree) dalam pemrograman adalah struktur data hierarkis yang terdiri dari simpul (nodes) dengan hubungan yang disebut sebagai tepi (edges). Setiap simpul dalam pohon memiliki satu simpul induk (parent node) kecuali simpul akar (root node) yang tidak memiliki induk. Pohon digunakan untuk merepresentasikan data dengan hubungan hierarkis seperti sistem file, struktur organisasi, dan ekspresi matematika.

Berikut adalah beberapa istilah yang umumnya digunakan dalam pohon:

- Predecessor: Node yang berada di atas node tertentu.
- Successor: Node yang berada di bawah node tertentu.
- Ancestor: Semua node yang terletak sebelum node tertentu dan berada pada jalur yang sama.
- Descendant: Semua node yang terletak sesudah node tertentu dan berada pada jalur yang sama.
- Parent: Predecessor satu level di atas suatu node.
- Child: Successor satu level di bawah suatu node.
- Sibling: Nodenode yang memiliki parent yang sama dengan suatu node.

- Subtree: Bagian dari pohon yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari pohon tersebut.
- Size: Banyaknya node dalam suatu pohon.
- Height: Banyaknya tingkatan/level dalam suatu pohon.
- Root: Satusatunya node khusus dalam pohon yang tidak memiliki predecessor.
- Leaf: Nodenode dalam pohon yang tidak memiliki successor.
- Degree: Banyaknya child yang dimiliki suatu node.

B. Guided

Guided 1

```
#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}};

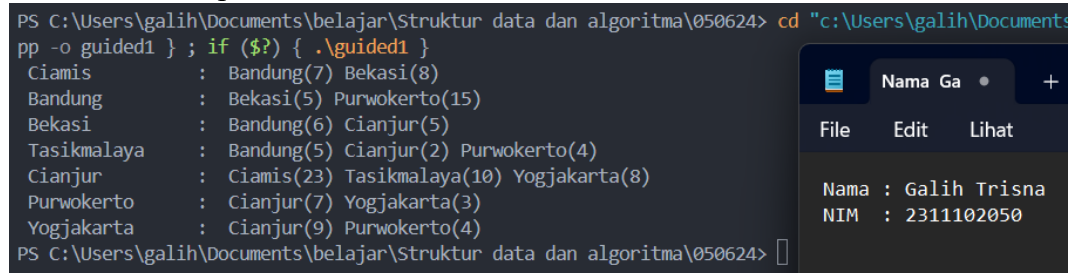
void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15) <<
simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "(" <<
busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}

int main()
{
    tampilGraph();
}
```

```
    return 0;
}
```

Screenshots Output

```
PS C:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624> cd "c:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624"
pp -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) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
PS C:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624>
```



Deskripsi:

Program di atas mendemonstrasikan penggunaan graf dengan menggunakan matriks (array dua dimensi) untuk menggambarkan keterhubungan antar simpul (verteks) dalam graf. Setiap baris dalam array memberikan informasi mengenai keterhubungan simpul tersebut dengan simpul lainnya. Saat dieksekusi, program akan melakukan looping pada setiap elemen dalam matriks. Pada setiap iterasi baris, program akan dimulai dengan menampilkan nama kota. Untuk setiap iterasi kolom, jika nilai elemen tidak sama dengan 0, program akan menampilkan simpul yang terhubung beserta bobot dari edge tersebut. Looping ini akan terus berjalan hingga elemen terakhir dalam matriks.

Guided 2

```
#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; // true
    else
        return 0; // 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();
}

```

Screenshots Output

```
PS C:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624> cd "c:\Users\

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kananA

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kananB

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kananE

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kananG

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
PS C:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624> 
```

 Nama Ga • +

FileEditLihat

Nama |: Galih Trisna
NIM : 2311102050

Ln 1, Col 6 | 38 karakter

Deskripsi:

Program di atas adalah contoh penggunaan struktur data pohon (tree). Dalam program ini, data pohon disimpan dalam bentuk node yang menyimpan nilai (value) dan pointer yang mengarah ke parent serta child (kanan & kiri) dari node tersebut. Saat dijalankan, program akan menginisialisasi pohon dengan membuat root terlebih dahulu menggunakan fungsi `buatNode()`. Selanjutnya, program akan membuat beberapa child (kanan & kiri) menggunakan fungsi `insertLeft()` untuk child kiri dan `insertRight()` untuk child kanan. Data node dalam program dapat diperbarui melalui fungsi `update()`. Fungsi `retrieve()` digunakan untuk menampilkan nilai dari node, sedangkan `find()` menampilkan informasi lebih rinci tentang node, termasuk parent, child, dan sibling.

Untuk menampilkan semua node dalam pohon, digunakan fungsi `preOrder()`, `inOrder()`, dan `postOrder()`, yang masing-masing memiliki urutan data yang berbeda. Selain itu, ada fungsi `characteristic()` yang memberikan informasi tentang ukuran, tinggi, dan rata-rata node dalam pohon. Fungsi ini memanggil dua fungsi lain, yaitu `size()` untuk mendapatkan ukuran pohon dan `height()` untuk mendapatkan tinggi pohon.

Untuk penghapusan node, terdapat tiga fungsi dalam program ini: `deleteTree()` untuk menghapus sebuah node, `deleteSub()` untuk menghapus subpohon (descendant dari node), dan `clear()` untuk menghapus seluruh pohon.

C. Unguided/Tugas

Unguided 1

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    int verCount;

    cout << "Masukkan jumlah simpul : ";
    cin >> verCount;

    string vertecies[verCount];
    int edgeValues[verCount][verCount];
    cout << "Masukkan nama simpul,\n";
    for (int i = 0; i < verCount; i++) {
        cout << "Simpul " << i + 1 << " : ";
        cin >> vertecies[i];
    }

    cout << "Masukkan bobot antar simpul,\n";
    for (int i = 0; i < verCount; i++) {
        for (int j = 0; j < verCount; j++) {
            cout << vertecies[i] << "->" << vertecies[j] << " : ";
            cin >> edgeValues[i][j];
        }
    }

    cout << endl << setw(10) << " ";
    for (int i = 0; i < verCount; i++) {
        cout << setw(10) << vertecies[i];
    }
    cout << endl;

    for (int i = 0; i < verCount; i++) {
        cout << setw(10) << vertecies[i];
        for (int j = 0; j < verCount; j++) {
            cout << setw(10) << edgeValues[i][j];
        }
    }
}
```

```

    }
    cout << endl;
}

return 0;
}

```

Screenshots Output

The screenshot shows a Windows command prompt window with the following text:

```

PS C:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624> c
+ unguided1.cpp -o unguided1 } ; if ($?) { .\unguided1 }
Masukkan jumlah simpul : 2
Masukkan nama simpul,
Simpul 1 : Jakarta
Simpul 2 : Bandung
Masukkan bobot antar simpul,
Jakarta->Jakarta : 0
Jakarta->Bandung : 12
Bandung->Jakarta : 12
Bandung->Bandung : 0

```

Below the command prompt, a Notepad++ window is open, displaying the output of the program:

```

Nama Ga
File Edit Lihat
Nama |: Galih Trisna
NIM  : 2311102050
Ln 1, Col 6 38 karakter 100%

```

At the bottom of the command prompt, a 3x3 adjacency matrix is displayed:

	Jakarta	Bandung
Jakarta	0	12
Bandung	12	0

The command prompt prompt is PS C:\Users\galih\Documents\belajar\Struktur data dan algoritma\050624>

Deskripsi

Program di atas adalah program graf yang menerima input dari pengguna. Pertama, program meminta input jumlah simpul dari graf. Kemudian, program melakukan looping untuk mendapatkan nama setiap simpul, di mana jumlah iterasi looping sesuai dengan jumlah simpul yang diinputkan. Setelah itu, program melakukan nested looping untuk memberikan nilai ke setiap edge graf, dengan jumlah iterasi yang dapat dirumuskan sebagai $n \times n$. Setelah semua edge diberi nilai, program akan menampilkan hasil input yang telah dimasukkan. Misalkan terdapat 3 daerah. Maka bobot yang diinput akan berjumlah 9, sesuai dengan ordo matriksnya yaitu 3×3 . Tampilan graf ini juga disajikan dalam bentuk matriks. Setelah tahap ini, program selesai dijalankan.

Unguided 2

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru, *current;
// Inisialisasi
void init()
{
    root = NULL;
}
// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; // true
    else
        return 0; // 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;
        current = root;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}
```

```

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat 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 Tidak ada tree, buat terlebih dahulu!!" <<
endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

```

```

    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Tidak ada tree, buat 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 Tidak ada tree, buat 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;
}

void showChild(Pohon *node)
{
    if (!root)
    {
        cout << "\n Tidak ada tree, buat terlebih dahulu!" <<
endl;
    }
    else
    {
        if (!node->left)
            cout << " Child kiri : (tidak ada child kiri)" <<
endl;
        else
            cout << " Child kiri : " << node->left->data <<
endl;
        if (!node->right)
            cout << " Child kanan : (tidak ada child kanan)" <<
endl;
        else
            cout << " Child kanan : " << node->right->data <<
endl;
    }
}

void showDescendant(Pohon *node = current)
{
    if (!root)
    {

```

```

        cout << "\n Tidak ada tree, buat terlebih dahulu!" <<
endl;
        return;
    }
    if (node != NULL)
    {
        if (node != current)
            cout << node->data << ", ";
        showDescendant(node->left);
        showDescendant(node->right);
    }
}
char inputData()
{
    char n;
    cout << "Masukkan data : ";
    cin >> n;
    return n;
}
void changeCurrent(char dest, Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Tidak ada tree, buat terlebih dahulu!" <<
endl;
    }
    else if (current->data == dest)
        return;
    else
    {
        if (node->data == dest)
        {
            Pohon *temp = current;
            current = node;
            cout << "\n Current node : " << temp->data << " ->
" << current->data << endl;
        }
        else
        {
            if (node->left != nullptr)
                changeCurrent(dest, node->left);
            if (node->right != nullptr)
                changeCurrent(dest, node->right);
        }
    }
}

```

```

    }
}
int main()
{
    string line(12, '=');
    int choice;

    while (true)
    {
        string options[] = {
            "Tampilkan node",
            "Tampilkan node (detail)",
            "Tampilkan child",
            "Tampilkan descendant",
            "Ukuran tree",
            "Tinggi tree",
            "characteristic tree",
            "Buat root",
            "Tambah child kiri",
            "Tambah child kanan",
            "Ganti current node",
            "Update data node",
            "Traversal Pre-Order",
            "Traversal Post-Order",
            "Traversal In-Order",
            "Hapus tree",
            "Hapus subtree",
            "Hapus node",
        };
        int optSize = sizeof(options) / sizeof(options[0]);
        cout << endl;
        cout << "#" << line << " PROGRAM TREE " << line << "#"
<< endl;
        for (int i = 0; i < optSize; i++)
        {
            cout << i + 1 << ". " << options[i] << endl;
        }
        cout << "0. Keluar\n";
        cout << "\nNode saat ini : ";
        if (isEmpty())
        {
            cout << "-\n";
        }
        else
    }
}

```

```

{
    cout << current->data << endl;
}
cout << "Pilih menu [0 - " << optSize << "] : ";
cin >> choice;

switch (choice)
{
case 0:
{
    cout << "\n\nKeluar dari aplikasi.\n";
    return 0;
    break;
}
case 1:
    retrieve(current);
    break;
case 2:
    find(current);
    break;
case 3:
    showChild(current);
    break;
case 4:
    showDescendant();
    cout << endl;
    break;
case 5:
    cout << "Ukuran tree : " << size() << endl;
    break;
case 6:
    cout << "Tinggi tree : " << height() << endl;
    break;
case 7:
    charateristic();
    break;
case 8:
    buatNode(inputData());
    break;
case 9:
    insertLeft(inputData(), current);
    break;
case 10:
    insertRight(inputData(), current);

```

```
        break;
    case 11:
        changeCurrent(inputData());
        break;
    case 12:
        update(inputData(), current);
        break;
    case 13:
        preOrder();
        break;
    case 14:
        postOrder();
        break;
    case 15:
        inOrder();
        break;
    case 16:
        clear();
        break;
    case 17:
        deleteSub(current);
        break;
    case 18:
        deleteTree(current);
        break;
    default:
        cout << "Input Tidak Benar !\n";
    }
}
```

Screenshots Output

#===== PROGRAM TREE =====#

1. Tampilkan node
2. Tampilkan node (detail)
3. Tampilkan child
4. Tampilkan descendant
5. Ukuran tree
6. Tinggi tree
7. characteristic tree
8. Buat root
9. Tambah child kiri
10. Tambah child kanan
11. Ganti current node
12. Update data node
13. Traversal Pre-Order
14. Traversal Post-Order
15. Traversal In-Order
16. Hapus tree
17. Hapus subtree
18. Hapus node
0. Keluar

Node saat ini : A

Pilih menu [0 - 18] : 1

Data node : A

Nama Ga

+ - □ ×

File Edit Lihat

Nama : Galih Trisna
NIM : 2311102050

Ln 1, Col 6 | 38 karakter | 100% | Window UTF-8

Node saat ini : B

Pilih menu [0 - 18] : 2

Data Node : B

Root : A

Parent : A

Sibling : C

Child Kiri : D

Child Kanan : E

Nama Ga

+ - □ ×

File Edit Lihat

Nama : Galih Trisna
NIM : 2311102050

Node saat ini : B

Pilih menu [0 - 18] : 3

Child kiri : D

Child kanan : E

Nama : Galih Trisna

NIM : 2311102050

0. Keluar

Node saat ini : B

Pilih menu [0 - 18] : 4

D, E,

#===== PROGRAM TREE =====#

Nama : Galih Trisna

NIM : 2311102050

Node saat ini : B Pilih menu [0 - 18] : 5 Ukuran tree : 5	Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 6 Tinggi tree : 3	Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 7 Size Tree : 5 Height Tree : 3 Average Node of Tree : 1	File Edit Lihat Nama : Galih Trisna NIM : 2311102050
Node saat ini : - Pilih menu [0 - 18] : 8 Masukkan data : A	File Edit Lihat Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 9 Masukkan data : D	Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 10 Masukkan data : E	Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 11 Masukkan data : A	Nama : Galih Trisna NIM : 2311102050
Node saat ini : E Pilih menu [0 - 18] : 12 Masukkan data : B	Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 13 A, B, D, E, C,	File Edit Lihat Nama : Galih Trisna NIM : 2311102050

Node saat ini : B Pilih menu [0 - 18] : 14 D, E, B, C, A,	File Edit Lihat Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 15 D, B, E, A, C,	Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 16 Pohon berhasil dihapus.	File Edit Lihat Nama : Galih Trisna NIM : 2311102050
Node saat ini : B Pilih menu [0 - 18] : 17 Node subtree B berhasil dihapus.	Nama : Galih Trisna NIM : 2311102050
Node saat ini : A Pilih menu [0 - 18] : 18	Nama : Galih Trisna NIM : 2311102050

Deskripsi

Program ini adalah modifikasi dari program unguided 2 yang digunakan untuk membuat dan menampilkan pohon (tree). Salah satu modifikasi yang dilakukan adalah penambahan menu dengan 18 opsi. Sebagian besar opsi menu ini dibuat untuk menghubungkan fungsi lama dengan menu. Selain itu, variabel baru bernama current ditambahkan untuk menyimpan data node saat ini, berfungsi sebagai pointer yang menentukan data mana yang akan diubah.

Beberapa fungsi baru juga ditambahkan, yaitu showChild(), showDescendant(), inputData(), dan changeCurrent(). Berikut rincian fungsi-fungsi tersebut:

1. showDescendant(): Menampilkan semua data di bawah node saat ini (current node).
2. showChild(): Menampilkan data child dari node saat ini.
3. changeCurrent(): Mengganti pointer yang menunjuk node saat ini, sehingga node tersebut dapat ditampilkan secara detail, diubah, atau dihapus.
4. inputData(): Menginputkan data karakter.

D. Kesimpulan

Graf adalah struktur data yang terdiri dari simpul-simpul (nodes) yang dihubungkan melalui sisi-sisi (edges). Beberapa jenis graf yang umum dibahas meliputi graf berarah, graf tak berarah, dan graf berbobot. Representasi graf biasanya menggunakan dua metode yaitu menggunakan array dua dimensi sebagai matriks dan menggunakan linked list.

Pohon adalah struktur data hierarkis di mana setiap simpul (node) dapat memiliki beberapa anak, tetapi hanya memiliki satu induk. Dalam praktikum ini juga dibahas jenis pohon biner, yaitu pohon yang setiap simpulnya memiliki maksimal dua anak.

E. Referensi

Asisten Praktikum. (2024). Modul IX : GRAPH DAN TREE

Bali, Daisma, 2023. Memahami Konsep Tree dalam Struktur Data Lengkap dengan Source Code Programnya. Diakses pada 11 Juni 2024, dari:

<https://daismabali.medium.com/memahami-konsep-tree-dalam-struktur-data-lengkap-dengan-source-code-programnya-acbd0a8733d6>

I., Dimetrio M., 2019. Struktur Data Tree. MID Koding. Diakses pada 11 Juni 2024, dari: <https://irvandimetrio21.home.blog/2019/07/05/struktur-data-tree/>

Zipur, Ahmad, 2023. 8 Struktur Data: Graph – Pengertian, Jenis, dan Algoritma. Diakses pada 11 Juni 2024 dari : <https://ahmadzipur.com/8-struktur-data-graph-pengertian-jenis-dan-algoritma/>

Parewa Labs , 2023. Graph Data Stucture. Diakses pada 11 Juni 2024 dari <https://www.programiz.com/dsa/graph>