

LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 9 GRAPH DAN TREE



Disusun Oleh :

Muhammad Rusdiyanto Asatman
2311102053

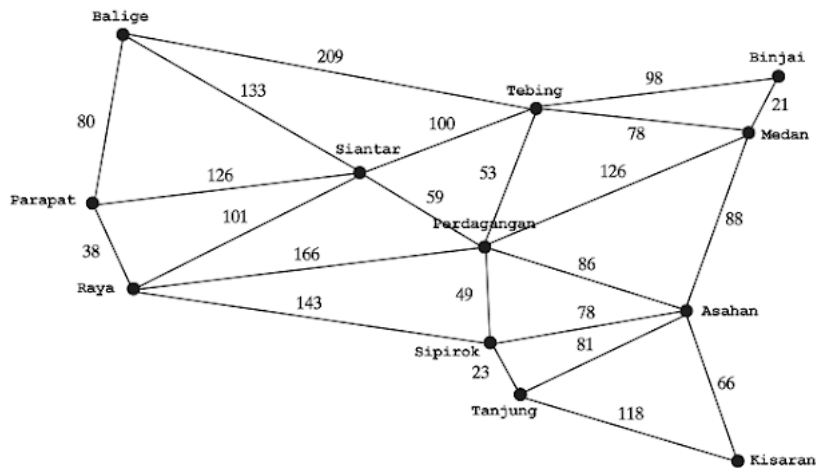
Dosen :

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

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

A. Dasar Teori

1. Graph



Gambar 1.0, Contoh Representasi Graf

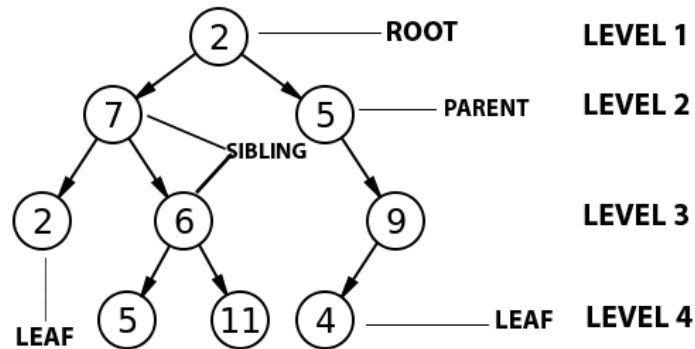
Graf dalam pemrograman adalah struktur data yang digunakan untuk merepresentasikan hubungan antara berbagai entitas atau objek. Sebuah graf terdiri dari simpul (nodes atau vertices) dan sisi (edges) yang menghubungkan pasangan simpul. Graf dapat digunakan untuk menyelesaikan berbagai jenis masalah dalam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

Terdapat beberapa jenis graf, yaitu :

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

Untuk mengimplementasikan graf dalam kode, biasanya ada dua cara umum untuk merepresentasikan graf yaitu menggunakan matriks ketetanggaan (adjacency matrix) atau daftar ketetanggaan (adjacency list).

2. Tree



Gambar 1.1, Representasi Tree (Pohon)

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

- 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.
- Descendant : seluruh node yang terletak sesudah 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 punya predecessor.
- Leaf : node-node dalam tree yang tak memiliki successor.
- Degree : banyaknya child yang dimiliki suatu node.

B. Guided

Guided 1

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

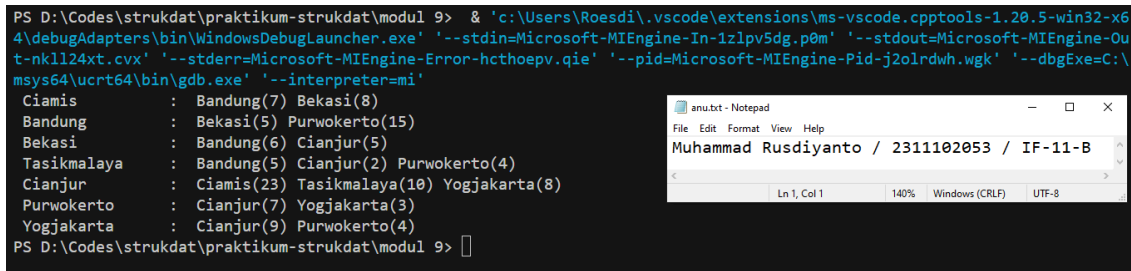
using namespace std;

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 D:\Codes\strukdat\praktikum-strukdat\modul 9> & 'c:\Users\Roesdi\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-1zlpv5dg.p0m' '--stdout=Microsoft-MIEngine-Out-nk1l24xt.cvx' '--stderr=Microsoft-MIEngine-Error-hcthoepv.qie' '--pid=Microsoft-MIEngine-Pid-j2olrdwh.wgk' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
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:\Codes\strukdat\praktikum-strukdat\modul 9> 
```

The screenshot shows a terminal window with the output of a graph traversal program. The output lists nodes and their connections with weights. To the right, a Notepad window is open, displaying the text: "Muhammad Rusdiyanto / 2311102053 / IF-11-B".

Deskripsi:

Program di atas adalah program yang mendemonstrasikan penggunaan graf. Dalam program ini, keterhubungan antar node (verteks) dalam graf digambarkan melalui matriks (array 2 dimensi). Setiap baris di dalam array memberikan informasi terkait keterhubungan node tersebut dengan node lain. Ketika di eksekusi, program akan looping ke setiap elemen dalam matriks. Untuk setiap iterasi baris matriks akan diawali oleh nama kota. Untuk iterasi kolom, jika nilai elemen tidak sama dengan 0, maka program akan menampilkan simpul yang terhubung dan bobot dari edge. Looping akan dijalankan sampai 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;
            }
        }
    }
}

```

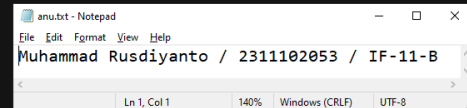
```

}
// 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();
}

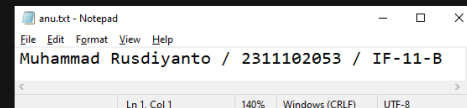
```

Screenshots Output

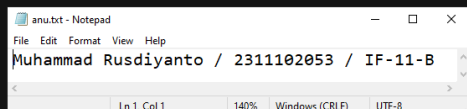
```
Node A berhasil dibuat menjadi root.  
Node B berhasil ditambahkan ke child kiri A  
Node C berhasil ditambahkan ke child kananA  
Node D berhasil ditambahkan ke child kiri B  
Node E berhasil ditambahkan ke child kananB  
Node F berhasil ditambahkan ke child kiri C  
Node G berhasil ditambahkan 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
```



Deskripsi:

Program di atas adalah program yang mendemonstrasikan penggunaan tree. Dalam program ini, data tree disimpan dalam bentuk node yang menyimpan nilai (value) dan pointer yang mengarah ke parent serta child (kanan & kiri) dari node tersebut. Ketika running, program akan menginisialisasi tree dengan membuat root dari tree terlebih dahulu dengan fungsi `buatNode()`. Setelah itu, program akan membuat beberapa child (kanan & kiri) dengan menggunakan `insertLeft()` untuk child kiri dan `insertRight()` untuk child kanan. Data node dalam program dapat diupdate melalui fungsi `update()`. Adapun fungsi `retrieve` untuk menampilkan nilai dari node dan `find()` untuk menampilkan data dari node dengan lebih rinci (informasi tentang parent, child,

sibling). Dalam menampilkan semua node dalam tree dapat dilakukan dengan menggunakan fungsi `preOrder()`, `inOrder()`, dan `postOrder()`. Perbedaan dari ketiga fungsi tersebut adalah urutan data yang ditampilkan. Adapula fungsi `characteristic()` yang digunakan untuk memberikan status terkait ukuran, tinggi, dan rata - rata node dalam tree. Fungsi tersebut memanggil 2 fungsi lain yaitu `size()` untuk mendapatkan ukuran tree dan `height()` untuk mendapatkan tinggi tree. Lalu untuk fungsi penghapusan dalam program ini terdiri dari 3, yaitu `deleteTree()` untuk menghapus node, `deleteSub()` untuk menghapus upapohon (descendant dari node), dan `clear()` untuk menghapus tree secara keseluruhan.

C. Unguided

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;
    }
}
```

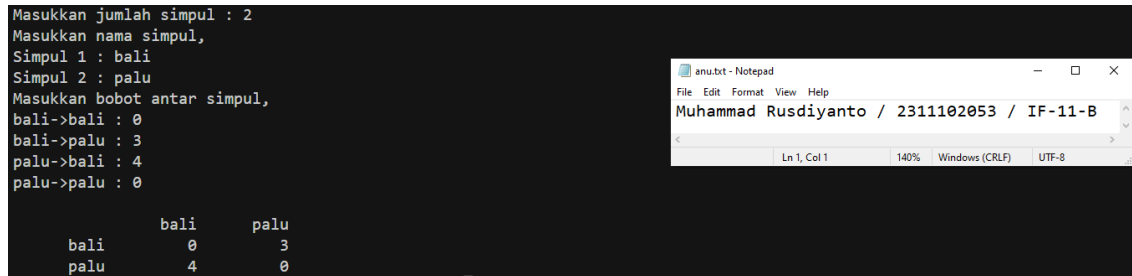
```

    }

    return 0;
}

```

Screenshots Output



```

Masukkan jumlah simpul : 2
Masukkan nama simpul,
Simpul 1 : bali
Simpul 2 : palu
Masukkan bobot antar simpul,
bali->bali : 0
bali->palu : 3
palu->bali : 4
palu->palu : 0

      bali    palu
bali    0      3
palu    4      0

```

Deskripsi:

Program di atas adalah sebuah program graf yang dapat menerima input dari pengguna. Pertama - tama program akan meminta input berupa jumlah simpul dari graf. Setelah itu, program melakukan looping untuk mendapatkan nama dari simpul. Banyaknya iterasi looping tadi akan menyesuaikan dengan jumlah simpul yang diinputkan. Setelah itu, program akan melakukan nested looping untuk memberikan nilai ke setiap edge dari graf. Banyaknya iterasi looping dapat dirumuskan sebagai $n \times n$ atau n^2 . Setelah semua edge diberikan nilai, maka program akan menampilkan hasil inputan tadi. Setelah itu, program selesai.

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 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;
}
void showChild(Pohon *node) {
    if (!root) {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else {
        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;
    }
}
void showDescendant(Pohon *node = current) {
    if (!root) {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return;
    }
    if (node != NULL) {
        if (node != current) cout << node->data << ", ";
        showDescendant(node->left);
        showDescendant(node->right);
    }
}
char inData() {
    char t;
    cout << "Masukkan data : ";
    cin >> t;
    return t;
}

```

```

void changeCurrent(char dest, Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree 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(15, '=');
    int choice;
    while(true) {
        string options[] = {
            "Tampilkan node", "Tampilkan node (detail)",
"Tampilkan child", "Tampilkan descendant", "Ukuran tree", "Tinggi
tree", "Karakteristik tree",
            "Buat root", "Tambah child (kiri)", "Tambah child
(kanan)",
            "Ganti `current node`", "Update data node",
            "Urutkan node (Pre-Order)", "Urutkan node (In-Order)",
"Urutkan node (Post-Order)",
            "Hapus tree", "Hapus sub-tree", "Hapus node",
        };
        int optSize = sizeof(options)/sizeof(options[0]);

        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;

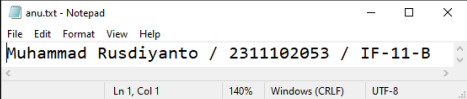
            case 7: charateristic(); break;
            case 8: buatNode(inData()); break;
            case 9: insertLeft(inData(), current); break;
            case 10: insertRight(inData(), current); break;
            case 11: changeCurrent(inData()); break;
            case 12: update(inData(), current); break;
            case 13: preOrder(); break;
            case 14: inOrder(); break;
            case 15: postOrder(); break;
            case 16: clear(); break;
            case 17: deleteSub(current); break;
            case 18: deleteTree(current); break;
            default: cout << "Tolong masukkan input yang
sesuai!\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. Karakteristik tree  
8. Buat root  
9. Tambah child (kiri)  
10. Tambah child (kanan)  
11. Ganti 'current node'  
12. Update data node  
13. Urutkan node (Pre-Order)  
14. Urutkan node (In-Order)  
15. Urutkan node (Post-Order)  
16. Hapus tree  
17. Hapus sub-tree  
18. Hapus node  
0. Keluar  
  
Node saat ini : -  
Pilih menu [0 - 18] : 
```



anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
0. Keluar  
  
Node saat ini : b  
Pilih menu [0 - 18] : 1  
  
Data node : b
```



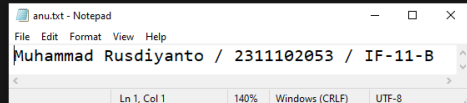
anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
Node saat ini : b  
Pilih menu [0 - 18] : 2  
  
Data Node : b  
Root : a  
Parent : a  
Sibling : c  
Child Kiri : d  
Child Kanan : e
```



anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
Node saat ini : b  
Pilih menu [0 - 18] : 3  
Child Kiri : d  
Child Kanan : e
```



anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
18. Hapus node  
0. Keluar  
  
Node saat ini : a  
Pilih menu [0 - 18] : 4  
b, d, e, c,
```



anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
0. Keluar  
  
Node saat ini : b  
Pilih menu [0 - 18] : 5  
Ukuran tree : 5
```



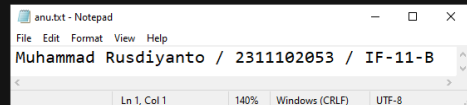
anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
0. Keluar  
  
Node saat ini : b  
Pilih menu [0 - 18] : 6  
Tinggi tree : 3
```



anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
Node saat ini : b  
Pilih menu [0 - 18] : 7  
  
Size Tree : 5  
Height Tree : 3  
Average Node of Tree : 1
```



anu.txt - Notepad
File Edit Format View Help
Muhammad Rusdiyanto / 2311102053 / IF-11-B
Ln 1, Col 1 140% Windows (CRLF) UTF-8

```
Node saat ini : -
Pilih menu [0 - 18] : 8
Masukkan data : a

Node a berhasil dibuat menjadi root.

Node saat ini : a
Pilih menu [0 - 18] : 9
Masukkan data : b

Node b berhasil ditambahkan ke child kiri a

Node saat ini : a
Pilih menu [0 - 18] : 10
Masukkan data : c

Node c berhasil ditambahkan ke child kanana

Node saat ini : a
Pilih menu [0 - 18] : 11
Masukkan data : b

Current node : a -> b

Node saat ini : b
Pilih menu [0 - 18] : 12
Masukkan data : L

Node b berhasil diubah menjadi L

0. Keluar

Node saat ini : L
Pilih menu [0 - 18] : 13
a, L, d, e, c, ===== Program Tree =====

0. Keluar

Node saat ini : L
Pilih menu [0 - 18] : 14
d, L, e, a, c, ===== Program Tree =====
1. Tampilkan node

Node saat ini : L
Pilih menu [0 - 18] : 15
d, e, L, c, a, ===== Program Tree =====

Node saat ini : L
Pilih menu [0 - 18] : 16

Pohon berhasil dihapus.

Node saat ini : L
Pilih menu [0 - 18] : 17

Node subtree L berhasil dihapus.
===== Program Tree =====

18. Hapus node
0. Keluar

Node saat ini : e
Pilih menu [0 - 18] : 18

Node saat ini : -
Pilih menu [0 - 18] : 0

Keluar dari aplikasi.
```

Deskripsi:

Program tersebut merupakan sebuah program untuk membuat dan menampilkan tree yang merupakan modifikasi dari program unguided 2. Modifikasi yang dilakukan

salah satunya adalah menambahkan menu. Untuk jumlah opsi dalam menu sendiri terdapat 18, sebagian besar dibuat untuk menghubungkan fungsi lama dengan menu. Dalam penghubungan ini, dibuat juga variabel baru yaitu `current` untuk menyimpan data node saat ini (semacam pointer yang menentukan data mana yang akan diubah saat itu). Disamping itu, ada juga beberapa fungsi baru seperti `showChild()`, `showDescendant()`, `inData()`, dan `changeCurrent()`. Berikut adalah uraian terkait rincian fungsi tersebut :

1. `showChild()` : digunakan untuk menampilkan data child dari node.
2. `showDescendant()` : fungsi untuk menampilkan data di bawah node saat ini (`current node`).
3. `inData()` : untuk menginputkan data char.
4. `changeCurrent()` : digunakan untuk mengganti pointer yang menunjuk node saat ini (node tersebut nantinya bisa ditampilkan detailnya, diubah, dihapus, dll.).

D. Kesimpulan

Graf dan tree adalah struktur data yang esensial dalam pemrograman, masing-masing digunakan untuk merepresentasikan dan mengelola hubungan antar elemen. Graf terdiri dari simpul (nodes) dan sisi (edges) yang menghubungkan pasangan simpul, digunakan dalam berbagai aplikasi seperti jaringan komputer, media sosial, dan peta. Tree, sejenis graf berarah yang tidak mengandung siklus, digunakan untuk struktur data hirarkis (yang memiliki level / tingkatan) seperti file system dan binary search tree (BST). Keduanya mendukung operasi seperti traversal, pencarian, penyisipan, dan penghapusan, namun tree memiliki struktur yang lebih teratur dengan satu simpul akar dan hierarki induk-anak yang jelas, sedangkan graf lebih fleksibel tanpa hirarki yang kaku. Memahami dan menerapkan graf dan tree dengan tepat dapat meningkatkan efisiensi dalam menyelesaikan berbagai masalah komputasi.

E. Referensi

Asisten Praktikum, “Modul 9 Graph dan Tree”

Zipur, Ahmad, 2023. 8 Struktur Data: Graph – Pengertian, Jenis, dan Algoritma. Tersedia di : <https://ahmadzipur.com/8-struktur-data-graph-pengertian-jenis-dan-algoritma/> [Diakses 5 Juni 2024].

Sianipar, Rismon H., 2016. Algoritma, Struktur Data, dan Pemrograman: Bab 12. Java Struktur Data dan Pemrograman GUI. Tersedia di : <https://rhsianipar.blogspot.com/2016/12/bab-12-java-struktur-data-dan.html> [Diakses 5 Juni 2024].

Bali, Daisma, 2023. Memahami Konsep Tree dalam Struktur Data Lengkap dengan Source Code Programnya. Medium. Tersedia di : <https://daismabali.medium.com/memahami-konsep-tree-dalam-struktur-data-lengkap-dengan-source-code-programnya-acbd0a8733d6> [Diakses 5 Juni 2024].

I., Dimetrio M., 2019. Struktur Data Tree. MID Koding. Tersedia di : <https://irvandimetrio21.home.blog/2019/07/05/struktur-data-tree/> [Diakses 5 Juni 2024].