

**LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA**

**MODUL IX
GRAPH DAN TREE**



Disusun Oleh :

Anisah Syifa Mustika Riyanto
2311102080

Dosen

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

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

A. Dasar Teori

Graph

Graf adalah kumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). *Graph* dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari *graph* adalah dengan menyatakan objek sebagai noktah, bulatan atau titik (*Vertex*), sedangkan hubungan antara objek dinyatakan dengan garis (*Edge*).

$$G = (V, E)$$

Dimana :

$G = \text{Graph}$

$V = \text{Simpul atau Vertex, atau Node, atau Titik}$

$E = \text{Busur atau Edge, atau arc}$

Berikut ini merupakan contoh program *graph*:

```
#include <stdfix.h>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
using namespace std;
int ordo[5][5];
void masukan(int a, int b, int c)
{
    for (int i = 1; i <= 4; i++)
    {
        for (int j = 1; j <= 4; j++)
            if (i == a && j == b)
            {
                ordo[a][b] = ordo[i][j];
                ordo[a][b] = c;
            }
    }
}
void tampilkan(int a, int b, int c)
{
    for (int i = 1; i <= 4; i++)
    {
        for (int j = 1; j <= 4; j++)
            cout << ordo[i][j] << " ";
        cout << endl;
    }
}
void inisialisasi()
{
    for (int i = 1; i <= 4; i++)
    {
```

```

        for (int j = 1; j <= 4; j++)
            ordo[i][j] = 0;
    }
}
void menu()
{
    cout << "-----MENU-----\n";
    cout << "1. Masukkan Data\n";
    cout << "2. Tampilkan\n";
    cout << "0. Keluar\n";
    cout << "Masukkan Pilihan Anda: ";
}
int main()
{
    inisialisasi();
    int a, b, c;
    int m;
    do
    {
        system("cls");
        menu();
        cin >> m;
        switch (m)
        {
            case 1:
                cout << "\nMasukkan Koordinat x : ";
                cin >> a;
                cout << "Masukkan Koordinat y : ";
                cin >> b;
                cout << "Masukkan Isi : ";
                cin >> c;
                if (a <= 4 && b <= 4)
                {
                    masukkan(a, b, c);
                }
                else
                {
                    cout << "\nIndeks harus kurang dari 4\n";
                }
                break;
                system("pause");
            case 2:
                system("cls");
                tampilkan(a, b, c);
                break;
                system("pause");
        }
        system("pause");
    } while (m != 0);
}

```

Tree

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya. Tree juga adalah suatu graph yang acyclic, simple, connected yang tidak mengandung loop.

Sebuah *binary search tree* (bst) adalah sebuah Tree biner yang boleh kosong, dan setiap *nodenya* harus memiliki *identifier/value*. *Value* pada semua *node* subTree sebelah kiiri adalah selalu lebih kecil dari *value* dari *root*, sedangkan *value* subTree di sebelah kanan adalah sama atau lebih besar dari *value* pada *root*, masing-masing subTree tersebut (kiri dan kanan) itu sendiri adalah juga *binary search tree*.

Struktur data bst sangat penting dalam struktur pencarian, misalkan dalam kasus pencarian dalam sebuah list, jika list sudah dalam keadaan terurut maka proses pencarian akan semakin cepat, jika kita menggunakan *list contigue* dan melakukan pencarian biner, akan tetapi jika kita ingin melakukan perubahan isi list (*insert* atau *delete*), menggunakan *list contigue* akan sangat lambat, karena prose *insert* dan *delete* dalam *list contigue* butuh memindahkan *linked-list*, yang untuk operasi *insert* atau *delete* tinggal mengatur- atur *pointer*, akan tetapi pada *n-linked list*, kita tidak bisa melakukan *pointer* sembarangan setiap saat, kecuali hanya satu kali dengan kata lain hanya secara *sequential*.

Berikut ini merupakan program *tree*:

```
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct Node
{
    int data;
    Node *kiri;
    Node *kanan;
};
int count;
void tambah(Node **root, int databaru)
{
    if ((*root) == NULL)
    {
        Node *baru;
        baru = new Node;
        baru->data = databaru;
        baru->kiri = NULL;
        baru->kanan = NULL;
        (*root) = baru;
        (*root)->kiri = NULL;
        (*root)->kanan = NULL;
        printf("Data telah Dimasukkan");
    }
    else if (databaru < (*root)->data)
```

```

        tambah(&(*root)->kiri, databaru);
    else if (databaru > (*root)->data)
        tambah(&(*root)->kanan, databaru);
    else if (databaru == (*root)->data)
        printf("Data sudah ada!!");
}
void preOrder(Node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preOrder(root->kiri);
        preOrder(root->kanan);
    }
}
void inOrder(Node *root)
{
    if (root != NULL)
    {
        inOrder(root->kiri);
        printf("%d ", root->data);
        inOrder(root->kanan);
    }
}
void postOrder(Node *root)
{
    if (root != NULL)
    {
        postOrder(root->kiri);
        postOrder(root->kanan);
        printf("%d ", root->data);
    }
}
void search(Node **root, int cari)
{
    if ((*root) == NULL)
    {
        printf("Maaf, Data tidak ditemukan!");
    }
    else if (cari < (*root)->data)
        search(&(*root)->kiri, cari);
    else if (cari > (*root)->data)
        search(&(*root)->kanan, cari);
    else if (cari == (*root)->data)
        printf("Data ditemukan!!!");
}
void hapus(Node **root, int del)
{
    if ((*root) == NULL)
    {
        printf("Data tidak ada!!");
    }
    else if (del < (*root)->data)
        hapus(&(*root)->kiri, del);
}

```

```

else if (del > (*root)->data)
    hapus(&(*root)->kanan, del);
else if (del == (*root)->data)
{
    (*root) = NULL;
    printf("Data telah Terhapus");
}
}
int main()
{
    int pil, cari, del;
    Node *Tree;
    Tree = NULL;
    do
    {
        int data;
        system("cls");
        printf("          PROGRAM TREE LANJUTAN      \n");
        printf("===== \n");
        printf("    1. Masukkan Data          \n");
        printf("    2. Transverse             \n");
        printf("    3. Cari                   \n");
        printf("    4. Hapus                  \n");
        printf("    5. Clear Data             \n");
        printf("    6. Keluar                 \n");
        printf("===== \n");
        printf("Masukkan Pilihan Anda : ");
        scanf("%d", &pil);
        switch (pil)
        {
            case 1:
                printf("Masukkan data baru : ");
                scanf("%d", &data);
                tambah(&Tree, data);
                break;
            case 2:
                printf("\nPreOrder : ");
                if (Tree != NULL)
                    preOrder(Tree);
                else
                    printf("Data masih kosong");
                printf("\ninOrder : ");
                if (Tree != NULL)
                    inOrder(Tree);
                else
                    printf("Data masih kosong");
                printf("\npostOrder : ");
                if (Tree != NULL)
                    postOrder(Tree);
                else
                    printf("Data masih kosong");
                break;
            case 3:
                printf("Cari data : ");

```

```

        scanf("%d", &cari);
        search(&Tree, cari);
        break;
    case 4:
        printf("Hapus data : ");
        scanf("%d", &del);
        hapus(&Tree, del);
        break;
    case 5:
        Tree = NULL;
        printf("Semua data telah terhapus");
        break;
    case 6:
        return 0;
    default:
        printf("Maaf, pilihan Anda Salah");
    }
    getch();
} while (pil != 7);
}

```

B. Guided

Guided 1

Program Graph

Source code:

```

#include <iostream>
#include <iomanip>

using namespace std;

string simpul[7] = {"Ciamis", "Bandung", "Bekasi", "Tasikmalaya",
"Cianjur", "Purwokerto", "Yogyakarta"};
int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

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

```

```

    }
    cout << endl;
}
}
int main()
{
    tampilGraph();
    return 0;
}

```

Screenshots Output:

```

PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC> & 'c:\Us
ptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '-
' '--stdout=Microsoft-MIEngine-Out-zejgszoh.bgf' '--stderr=Microsoft-M
oft-MIEngine-Pid-fqu2byva.0os' '--dbgExe=C:\Program Files\CodeBlocks\M
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:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC>

```

Anisah Syifa Mustika Riyanto
2311102080
IF-11-B

Deskripsi:

Program ini menciptakan sebuah graf berarah dengan simpul yang mewakili kota-kota dan busur yang mewakili jarak antar kota. Nama kota disimpan dalam array "simpul" dan jarak antar kota disimpan dalam array "busur". Dengan menggunakan fungsi "main", fungsi "tampilGraph" menampilkan informasi graf dalam bentuk yang mudah dibaca, dengan setiap baris menunjukkan kota asal, kota tujuan, dan jarak, jika ada koneksi antar kota.

Guided 2

Program Tree

Source code:

```

#include <iostream>
#include <iomanip>

using namespace std;

struct Tree
{
    char data;
    Tree *left, *right, *parent;
};

Tree *root, *baru;

void init()
{
    root = NULL;
}

```



```

}

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

void buatNode(char data)
{
    if (isEmpty())
    {
        root = new Tree();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat sebagai
root."
        << endl;
    }
    else
    {
        cout << "\n Tree sudah ada!" << endl;
    }
}

Tree *insertLeft(char data, Tree *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalo ada
            cout << "\n Node " << node->data << " sudah ada child
kiri !" << endl;
            return NULL;
        }
        else
        {
            // kalo gada
            Tree *baru = new Tree();
            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
Tree *insertRight(char data, Tree *node)
{

```

```

        if (isEmpty())
        {
            cout << "\n Buat tree terlebih dahulu!" << endl;
            return NULL;
        }
        else
        {
            // cek apakah child kanan ada atau tidak
            if (node->right != NULL)
            {
                // kalo ada
                cout << "\n Node " << node->data << " sudah ada child
kanan !" << endl;
                return NULL;
            }
            else
            {
                // kalo gada
                Tree *baru = new Tree();
                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;
            }
        }
    }

void update(char data, Tree *node)
{
    if (isEmpty())
    {
        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;
        }
    }
}

void retrieve(Tree *node)
{
    if (isEmpty())
    {
        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;
    }
}

}

void find(Tree *node)
{
    if (isEmpty())
    {
        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;
        }
    }
}

```

```

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

// inOrder
void inOrder(Tree *node = root)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Tree *node = root)
{
    if (isEmpty())
    {
        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(Tree *node)
{
    if (isEmpty())
    {

```

```

        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(Tree *node)
{
    if (isEmpty())
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

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

// Cek Size Tree
int size(Tree *node = root)
{
    if (isEmpty())
    {
        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(Tree *node = root)
{
    if (isEmpty())
    {
        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 characteristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() <<
endl;
}

int main()
{
    buatNode('A');
    Tree *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);
        characteristic();

        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;
    }

```

Screenshots Output:

```

PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC> & 'c:\Users\hp151\.vscode\extensions\ms-vsco
ptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-qcr2voy
' '--stdout=Microsoft-MIEngine-Out-uqfzvxi3.dse' '--stderr=Microsoft-MIEngine-Error-zwtb12e4.yqc' '--pid=M
oft-MIEngine-Pid-xxhj5mn1.shs' '--dbgExe=C:\Program Files\CodeBlocks\MinGW\bin\gdb.exe' '--interpreter=mi'

Node A berhasil dibuat sebagai 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

```

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

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

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,

PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC>
```



Deskripsi:

Program ini mengimplementasikan struktur data Tree biner dan berbagai operasi terkait, seperti penambahan simpul, pengubahan nilai simpul, pengambilan data simpul, dan pencarian simpul. Sementara fungsi "buatNode" membuat simpul akar, fungsi "masukkan Kiri" dan "masukkan Kanan" menambahkan simpul anak kiri dan kanan. Selain itu, program ini memiliki kemampuan menelusuri Tree dengan metode pre-order, in-order, dan post-order, serta kemampuan untuk menghapus simpul dan subtree. Selain itu, ada fungsi yang menampilkan karakteristik Tree dan menghitung ukuran dan tinggi Tree. Mula-mula, program menampilkan struktur Tree dalam berbagai urutan penelusuran setelah membuat beberapa simpul dan menambahkannya ke dalam Tree. Setelah itu, data simpul ditampilkan dan diubah.

C. Unguided

Unguided 1

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

Output Program:

```
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukan bobot antar simpul
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

      BALI    PALU
BALI   0      3
PALU   4      0
```

Source code:


```

#include <iostream>
#include <iomanip>
#include <vector>
#include <string>

using namespace std;

int main()
{
    int Anisah_2311102080;

    cout << "Silakan masukkan jumlah simpul: ";
    cin >> Anisah_2311102080;

    vector<string> simpul(Anisah_2311102080);

    vector<vector<int>> busur(Anisah_2311102080,
vector<int>(Anisah_2311102080, 0));

    cout << "Silakan masukkan nama simpul " << endl;
    for (int i = 0; i < Anisah_2311102080; i++)
    {
        cout << "Simpul ke-" << (i + 1) << ": ";
        cin >> simpul[i];
    }

    cout << "Silakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < Anisah_2311102080; i++)
    {
        for (int j = 0; j < Anisah_2311102080; j++)
        {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> busur[i][j];
        }
    }

    cout << endl;

    cout << setw(7) << " ";

    for (int i = 0; i < Anisah_2311102080; i++)
    {
        cout << setw(8) << simpul[i];
    }

    cout << endl;

    for (int i = 0; i < Anisah_2311102080; i++)

```

```

    {
        cout << setw(7) << simpul[i];
        for (int j = 0; j < Anisah_2311102080; j++)
        {
            cout << setw(8) << busur[i][j];
        }
        cout << endl;
    }
}

```

Screenshoot Output:

```

PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC> & 'c:\Users\hp151\.vscode\extensions\ms-vsco
gAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-fdedzgta.x1z' '--stdout=Microsoft-M
rr=Microsoft-MIEngine-Error-aewnvzss.4zx' '--pid=Microsoft-MIEngine-Pid-1n5ibi3b.utn' '--dbgExe=C:\Program
xe' '--interpreter=mi'
Silakan masukkan jumlah simpul: 2
Silakan masukkan nama simpul
Simpul ke-1: BALI
Simpul ke-2: PALU
Silakan masukkan bobot antar simpul
BALI --> BALI = 0
BALI --> PALU = 3
PALU --> BALI = 4
PALU --> PALU = 0

          BALI    PALU
BALI      0       3
PALU      4       0
PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC>

```

Deskripsi:

Program ini meminta pengguna memasukkan jumlah simpul, nama setiap simpul, dan bobot antar simpul ke dalam sebuah graf, lalu menampilkan matriks bobot yang menunjukkan hubungan antar simpul. Proses ini dimulai dengan meminta pengguna memasukkan jumlah simpul dan kemudian menyimpan nama-nama simpul dalam sebuah vektor; setelah itu, program meminta pengguna memasukkan bobot untuk setiap pasangan simpul dan menyimpannya dalam matriks dua dimensi.

Unguided 2

Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinput kan!

Source code:

```

#include <iostream>
#include <queue>
using namespace std;

```

```

// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;

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

// Cek Node
int isEmpty()
{
    return (root == NULL);
}

// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty())
    {
        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;
    }
}

// Cari Node Berdasarkan Data
Pohon *findNode(Pohon *node, char data)
{
    if (node == NULL)
        return NULL;
    if (node->data == data)
        return node;
    Pohon *foundNode = findNode(node->left, data);
    if (foundNode == NULL)
        foundNode = findNode(node->right, data);
    return foundNode;
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        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 (isEmpty())
    {
        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())
    {
        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)
{
    if (node != NULL)
    {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

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

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

// Hapus Node Tree
void deleteTree(Pohon *node)
{

```

```

    if (node != NULL)
    {
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root)
        {
            delete root;
            root = NULL;
        }
        else
        {
            delete node;
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (node != NULL)
    {
        deleteTree(node->left);
        deleteTree(node->right);
        node->left = NULL;
        node->right = NULL;
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Hapus Tree
void clear()
{
    deleteTree(root);
    cout << "\n Pohon berhasil dihapus." << endl;
}

// Cek Size Tree
int size(Pohon *node)
{
    if (node == NULL)
    {
        return 0;
    }
    else
    {
        return 1 + size(node->left) + size(node->right);
    }
}

// Cek Height Level Tree
int height(Pohon *node)
{
    if (node == NULL)
    {
        return 0;
    }
    else
    {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);

```

```

        return max(heightKiri, heightKanan) + 1;
    }
}

// Karakteristik Tree
void characteristic()
{
    cout << "\n Size Tree : " << size(root) << endl;
    cout << " Height Tree : " << height(root) << endl;
    cout << " Average Node of Tree : " << (height(root) == 0 ? 0 :
size(root) / height(root)) << endl;
}

// Menampilkan Child Node
void showChildren(Pohon *node)
{
    if (node)
    {
        if (node->left)
            cout << " Child Kiri: " << node->left->data << endl;
        else
            cout << " Child Kiri: (tidak memiliki Child kiri)" <<
endl;
        if (node->right)
            cout << " Child Kanan: " << node->right->data << endl;
        else
            cout << " Child Kanan: (tidak memiliki Child kanan)"
<< endl;
    }
}

// Menampilkan Descendants Node
void showDescendants(Pohon *node)
{
    if (node)
    {
        cout << " Descendants of Node " << node->data << ": ";
        preOrder(node);
        cout << endl;
    }
}

void menu()
{
    int pilihan;
    char data;
    char parentData_Anisah_2311102080;
    Pohon *temp = nullptr;
    do
    {
        cout << "\nMENU:\n";
        cout << "1. Buat Node Root\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
        cout << "4. Update Node\n";
        cout << "5. Retrieve Node\n";
        cout << "6. Find Node\n";
        cout << "7. Tampilkan PreOrder\n";
        cout << "8. Tampilkan InOrder\n";
        cout << "9. Tampilkan PostOrder\n";
        cout << "10. Tampilkan Characteristic\n";
    }
}

```



```

cout << "11. Hapus SubTree\n";
cout << "12. Hapus Tree\n";
cout << "13. Tampilkan Children\n";
cout << "14. Tampilkan Descendants\n";
cout << "0. Keluar\n";
cout << "Masukkan pilihan: ";
cin >> pilihan;
switch (pilihan)
{
case 1:
    if (isEmpty())
    {
        cout << "Masukkan data root: ";
        cin >> data;
        buatNode(data);
    }
    else
    {
        cout << "\n Root sudah ada!" << endl;
    }
    break;
case 2:
    if (!isEmpty())
    {
        cout << "Masukkan data node kiri: ";
        cin >> data;
        cout << "Masukkan data parent: ";
        cin >> parentData_Anisah_2311102080;
        temp = findNode(root,
parentData_Anisah_2311102080);
        insertLeft(data, temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 3:
    if (!isEmpty())
    {
        cout << "Masukkan data node kanan: ";
        cin >> data;
        cout << "Masukkan data parent: ";
        cin >> parentData_Anisah_2311102080;
        temp = findNode(root,
parentData_Anisah_2311102080);
        insertRight(data, temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 4:
    if (!isEmpty())
    {
        cout << "Masukkan data baru: ";
        cin >> data;
        cout << "Masukkan data node yang akan diupdate: ";
        cin >> parentData_Anisah_2311102080;
        temp = findNode(root,

```

```

parentData_Anisah_2311102080);
        update(data, temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 5:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang akan dilihat: ";
        cin >> parentData_Anisah_2311102080;
        temp = findNode(root,
parentData_Anisah_2311102080);
        retrieve(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 6:
    if (!isEmpty())
    {
        cout << "Masukkan data node yang akan dicari: ";
        cin >> parentData_Anisah_2311102080;
        temp = findNode(root,
parentData_Anisah_2311102080);
        find(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 7:
    if (!isEmpty())
    {
        cout << "\n PreOrder :" << endl;
        preOrder(root);
        cout << "\n"
            << endl;
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 8:
    if (!isEmpty())
    {
        cout << "\n InOrder :" << endl;
        inOrder(root);
        cout << "\n"
            << endl;
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }

```

```

        break;
    case 9:
        if (!isEmpty())
        {
            cout << "\n PostOrder :" << endl;
            postOrder(root);
            cout << "\n"
                << endl;
        }
        else
        {
            cout << "\n Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 10:
        if (!isEmpty())
        {
            characteristic();
        }
        else
        {
            cout << "\n Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 11:
        if (!isEmpty())
        {
            cout << "Masukkan data node yang subtreenya akan
dihapus: ";
            cin >> parentData_Anisah_2311102080;
            temp = findNode(root,
parentData_Anisah_2311102080);
            deleteSub(temp);
        }
        else
        {
            cout << "\n Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 12:
        clear();
        break;
    case 13:
        if (!isEmpty())
        {
            cout << "Masukkan data node yang akan ditampilkan
childnya: ";
            cin >> parentData_Anisah_2311102080;
            temp = findNode(root,
parentData_Anisah_2311102080);
            showChildren(temp);
        }
        else
        {
            cout << "\n Buat tree terlebih dahulu!" << endl;
        }
        break;
    case 14:
        if (!isEmpty())
        {
            cout << "Masukkan data node yang akan ditampilkan

```

```

descendantnya: ";
        cin >> parentData_Anisah_2311102080;
        temp = findNode(root,
parentData_Anisah_2311102080);
        showDescendants(temp);
    }
    else
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    break;
case 0:
    cout << "\n Anda telah keluar dari program!0" << endl;
    break;
default:
    cout << "\n Pilihan tidak valid!" << endl;
}
} while (pilihan != 0);
}

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

```

Screenshots Output:

```

PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC> & 'c:\Users\hp151\.vscode\extensions-vscode.cpptools-1.20.5-win32-x64\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-btamhwgt.tfb' '--stdout=Microsoft-MIEngine-Out-koh14w.tfb' '--stderr=Microsoft-MIEngine-Error-zrgqp2bv.oe2' '--pid=Microsoft-MIEngine-Pid-0wfltnyw.lce' '--dbgExe=C:\Program Files\CodeBlocks\bin\gdb.exe' '--interpreter=mi'

```

```

MENU:
3. Tambah Node Kanan
4. Update Node
5. Retrieve Node
6. Find Node
7. Tampilkan PreOrder
8. Tampilkan InOrder
9. Tampilkan PostOrder
10. Tampilkan Characteristic
11. Hapus SubTree
12. Hapus Tree
13. Tampilkan Children
14. Tampilkan Descendants
0. Keluar
Masukkan pilihan: 1
Masukkan data root: A

Node A berhasil dibuat menjadi root.

```

```

File Edit View

Anisah Syifa Mustika Riyanto
2311102080
IF-11-B

Ln 1, Col 4 | 47 characters | 100% | W

```

```

Masukkan pilihan: 2
Masukkan data node kiri: B
Masukkan data parent: A

Node B berhasil ditambahkan ke child kiri A

```

```

Anisah Syifa Mustika Riyanto
2311102080
IF-11-B

```

```

Masukkan pilihan: 3
Masukkan data node kanan: C
Masukkan data parent: A

Node C berhasil ditambahkan ke child kanan A

```

```

Anisah Syifa Mustika Riyanto
2311102080
IF-11-B

```

```

Masukkan pilihan: 8

InOrder :
B, A, C,

```

```

Anisah Syifa Mustika Riyanto
2311102080
IF-11-B

```

Masukkan pilihan: 9	Anisah Syifa Mustika Riyanto 2311102080 IF-11-B
PostOrder : B, C, A,	
Masukkan pilihan: 10	Anisah Syifa Mustika Riyanto 2311102080 IF-11-B
Size Tree : 3 Height Tree : 2 Average Node of Tree : 1	
Masukkan pilihan: 14 Masukkan data node yang akan ditampilkan descendantnya: A Descendants of Node A: A, B, C,	Anisah Syifa Mustika Riyanto 2311102080 IF-11-B
Masukkan pilihan: 13 Masukkan data node yang akan ditampilkan childnya: A Child Kiri: B Child Kanan: C	Anisah Syifa Mustika Riyanto 2311102080 IF-11-B
Masukkan pilihan: 0 Anda telah keluar dari program! PS D:\Huru Hara Semester 2\Praktikum Strukdat\Praktikum VSC> █	Anisah Syifa Mustika Riyanto 2311102080 IF-11-B

Deskripsi:

Program ini mengimplementasikan pohon biner. Program ini memungkinkan pengguna untuk membuat pohon biner, menambahkan node baru sebagai anak kiri atau kanan dari node tertentu, mengubah data node, menemukan node berdasarkan data, serta menampilkan traversals (preOrder, inOrder, dan postOrder) dan karakteristik pohon seperti ukuran dan tinggi. Selain itu, program juga menyediakan opsi untuk menghapus subtree atau pohon secara keseluruhan, serta menampilkan anak-anak atau keturunan dari suatu node tertentu dalam pohon. Pengguna dapat menjelajahi dan memanipulasi struktur pohon.

D. Kesimpulan

Berikut adalah kesimpulan mengenai materi Graph dan Tree:

1. Struktur Data Dasar: Graf dan pohon adalah struktur data untuk mewakili hubungan antar elemen, dengan pohon sebagai jenis graf tanpa siklus dan satu simpul akar.
2. Graf diimplementasikan menggunakan matriks atau daftar ketetanggaan, sedangkan pohon menggunakan struktur simpul dengan pointer ke anak-anaknya.
3. Operasi graf meliputi penambahan simpul, busur, dan penelusuran (DFS, BFS). Pada pohon, operasi mencakup penambahan, penghapusan, dan penelusuran simpul (in-order, pre-order, post-order).
4. Pohon biner memiliki simpul dengan paling banyak dua anak. BST adalah pohon biner yang mempermudah pencarian, penambahan, dan penghapusan dengan aturan elemen kiri lebih kecil dari induk, dan elemen kanan lebih besar.
5. Graf digunakan dalam jaringan komputer dan optimasi rute, sementara pohon digunakan dalam struktur data seperti heap dan trie, serta representasi hierarki sistem file dan database.

E. Referensi

Ahmaddhadarii77. (2019) Graph Graf dan Tree Algoritma. Diakses 7 Juni 2024, dari <https://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-Tree-algoritma.html>

Alia, P. A., & S ST, M. T. (2023). Dasar-Dasar Pemrograman.

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++* (Vol. 1). CV. AA. RIZKY.

Siahaan, V., & Sianipar, R. H. (2020). *Mudah Menguasai C++ Untuk Programmer*. BALIGE PUBLISHING. Huda, A., Ardi, N., & Muabi, A. (2021). *Pengantar Coding Berbasis C/C++*. UNP PRESS.

Zhang, J., Wang, L., Lee, R. K. W., Bin, Y., Wang, Y., Shao, J., & Lim, E. P. (2020, July). Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 3928-3937).