LAPORAN PRAKTIKUM

MODUL IX
GRAPH DAN TREE

Disusun oleh:
Anita Nurazizah Agussalim
NIM: 2311102017

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng.

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

BAB I

TUJUAN PRAKTIKUM

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

BAB II

DASAR TEORI

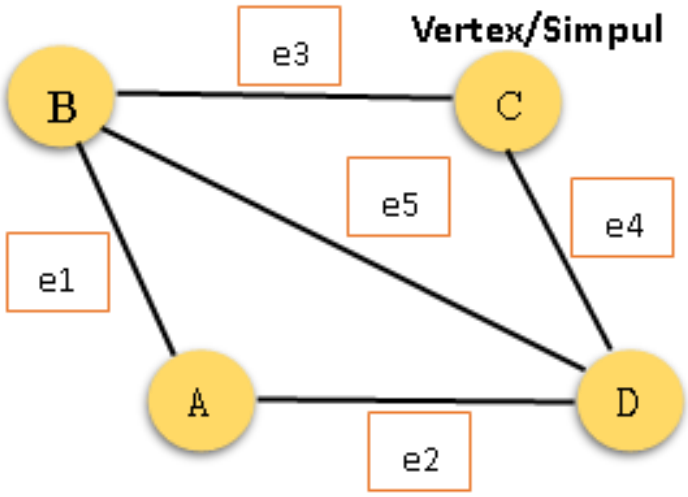
1. Graph

   Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

   $$G = (V, E)$$

   Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde.

   Dapat digambarkan:



Gambar 1 Contoh Graph

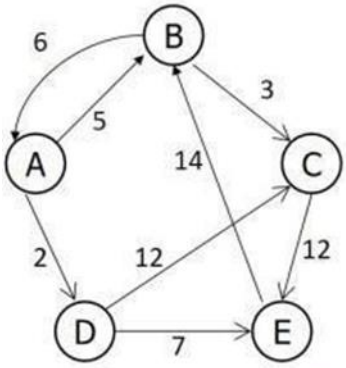
Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph:

- Directed Graph
- Undirected Graph
- Weight Graph

a. **Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
b. **Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
c. **Weigth Graph** : Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks

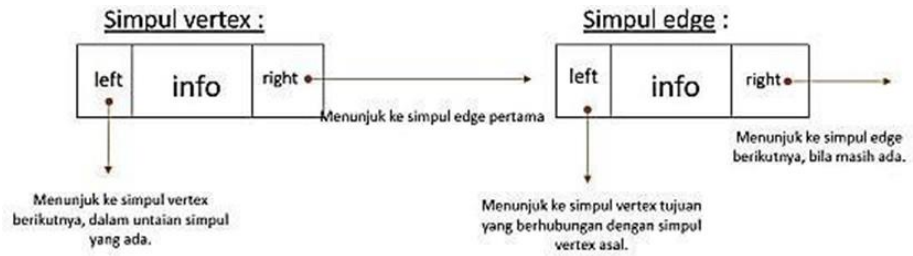


Ke →

Dari		A 0	B 1	C 2	D 3	E 4
A	0	0	5	0	2	0
B	1	6	0	3	0	0
C	2	0	0	0	0	9
D	3	0	0	12	0	7
E	4	0	14	0	0	0

Gambar 2 Representasi Graph dengan Matriks

Representasi dengan Linked List

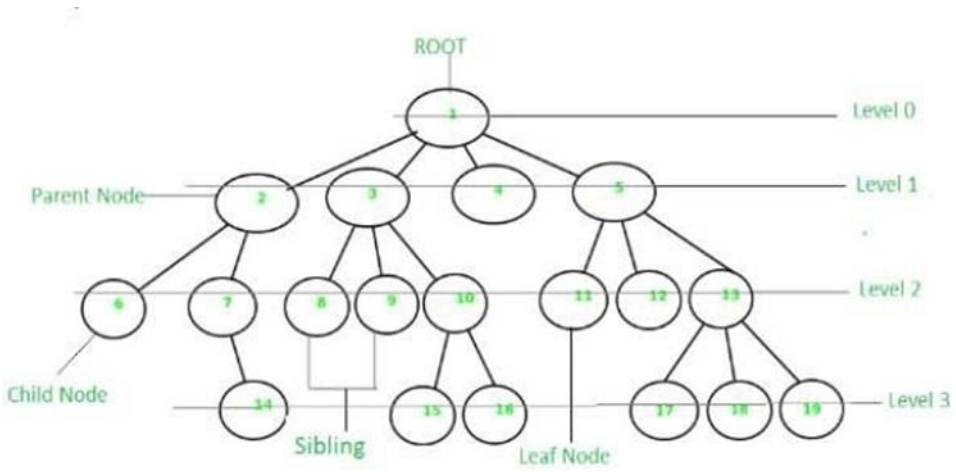


Gambar 3 Representasi Graph dengan Linked List

Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.

2. Tree atau Pohon

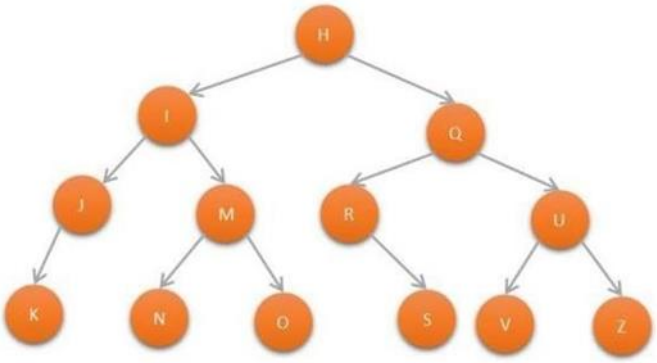
   Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



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

Tabel 1 Terminologi dalam Struktur Data Tree

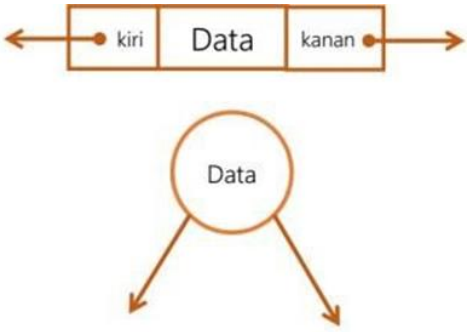
Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



Gambar 1 Struktur Data Binary Tree

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

a. Create: digunakan untuk membentuk binary tree baru yang masih kosong.
b. Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
c. isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
d. Insert: digunakan untuk memasukkan sebuah node kedalam tree.

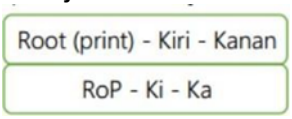
e. Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
f. Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
g. Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
h. Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
i. Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
j. Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

1) Pre-Order

Penelusuran secara pre-order memiliki alur:

a) Cetak data pada simpul root
b) Secara rekursif mencetak seluruh data pada subpohon kiri
c) Secara rekursif mencetak seluruh data pada subpohon kanan.

Dapat kita turunkan rumus penelusuran menjadi:



2) In-Order

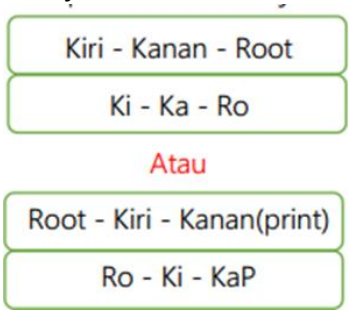
Penelusuran secara in-order memiliki alur:

a) Secara rekursif mencetak seluruh data pada subpohon kiri
b) Cetak data pada root
c) Secara rekursif mencetak seluruh data pada subpohon kanan.
Dapat kita turunkan rumus penelusuran menjadi:

3) Post Order

Penelusuran secara in-order memiliki alur:

a) Secara rekursif mencetak seluruh data pada subpohon kiri
b) Secara rekursif mencetak seluruh data pada subpohon kanan
c) Cetak data pada root
Dapat kita turunkan rumus penelusuran menjadi:



BAB III

GUIDED

1. Program Graph

Source code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                    "Bandung",
                    "Bekasi",
                    "Tasikmalaya",
                    "Cianjur",
                    "Purwokerto",
                    "Yogjakarta"};
int busur[7][7] =
    {
        {0, 7, 8, 0, 0, 0, 0},
        {0, 0, 5, 0, 0, 15, 0},
        {0, 6, 0, 0, 5, 0, 0},
        {0, 5, 0, 0, 2, 4, 0},
        {23, 0, 0, 10, 0, 0, 8},
        {0, 0, 0, 0, 7, 0, 3},
        {0, 0, 0, 0, 9, 4, 0}};
void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15)
             << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "("
                     << busur[baris][kolom] << ")";
            }
        }
        cout << endl;
```

```
    }
}

int main()
{
    tampilGraph();
    return 0;
}
```

Screenshot Output

```
 Ciamis          :  Bandung(7) Bekasi(8)
 Bandung         :  Bekasi(5) Purwokerto(15)
 Bekasi          :  Bandung(6) Cianjur(5)
 Tasikmalaya     :  Bandung(5) Cianjur(2) Purwokerto(4)
 Cianjur         :  Ciamis(23) Tasikmalaya(10) Yogjakarta(8)
 Purwokerto      :  Cianjur(7) Yogjakarta(3)
 Yogjakarta      :  Cianjur(9) Purwokerto(4)
```

Deskripsi program

Program di atas mendefinisikan sebuah grafik yang merepresentasikan hubungan antar kota di Jawa Barat dan sekitarnya menggunakan simpul dan busur. Simpul-simpulnya adalah kota-kota seperti Ciamis, Bandung, Bekasi, Tasikmalaya, Cianjur, Purwokerto, dan Yogyakarta. Matriks busur menyimpan jarak atau bobot antara setiap pasangan kota, dengan nilai nol menunjukkan tidak adanya hubungan langsung antara dua kota tersebut. Fungsi 'tampilGraph' digunakan untuk menampilkan grafik tersebut dalam bentuk yang mudah dibaca, di mana setiap kota (simpul) dicetak bersama dengan kota-kota yang terhubung dengannya dan jarak (bobot) yang terkait. Program utama memanggil fungsi ini untuk menampilkan grafik ketika dijalankan.

2. Program Tree

SOURCE CODE

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE

// Deklarasi Pohon
```

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

Pohon *root, *baru;

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

// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1;
    else
        return 0;
    // true
    // false
}

// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat menjadi root."
             << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
```

```
    }
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kiri!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke
child kiri " << baru->parent->data << endl;
                return baru;
        }
    }
}

// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
```

```
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kanan!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke
child kanan " << baru->parent->data << endl;
                return baru;
        }
    }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
```

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

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
```

```
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;

            if (!node->parent)
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->data << endl;

            if (node->parent != NULL && node->parent->left != node &&
                node->parent->right == node)

                cout << " Sibling : " << node->parent->left->data <<
endl;

            else if (node->parent != NULL && node->parent->right !=
node &&
                     node->parent->left == node)

                cout << " Sibling : " << node->parent->right->data <<
endl;
            else
                cout << " Sibling : (tidak punya sibling)" << endl;

            if (!node->left)
                cout << " Child Kiri : (tidak punya Child kiri)" <<
endl;
            else
                cout << " Child Kiri : " << node->left->data << endl;

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

```
// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {

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

// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {

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

// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
```

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

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

            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}

// Hapus SubTree
```

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

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

// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {

        if (!node)
        {
            return 0;
        }
        else
```

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

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);

            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
```

```
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

int main()
{
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;

    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);

    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
         << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
         << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
         << endl;

    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
```

```
    preOrder();
    cout << "\n"
         << endl;
    charateristic();
}
```

SCREENSHOT PROGRAM

```
Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kiri E

Node H berhasil ditambahkan ke child kanan E

Node I berhasil ditambahkan ke child kiri G

Node J berhasil ditambahkan ke child kanan G

Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Data node : C
```

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

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

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

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


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

Node subtree E berhasil dihapus.

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


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

DESKRIPSI PROGRAM

Program di atas adalah sebuah implementasi sederhana dari struktur data pohon biner menggunakan bahasa pemrograman C++. Program ini memungkinkan pengguna untuk membuat pohon biner, menambahkan node baru baik ke kiri maupun kanan dari sebuah node, mengubah data pada sebuah node, menampilkan informasi tentang sebuah node seperti data, parent, sibling, child kiri, dan child kanan, serta melakukan penelusuran (traversal) preOrder, inOrder, dan postOrder pada pohon. Selain itu, program ini juga dapat menghitung ukuran (size) dan tinggi (height) pohon, serta menampilkan karakteristik seperti jumlah rata-rata node. Program juga memiliki fungsi untuk menghapus subtree tertentu atau seluruh pohon.

LATIHAN KELAS - UNGUIDED

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

Source code

```
      #include <iostream>
#include <vector>
#include <string>

using namespace std;

void printMatrix(const vector<vector<int>>& matrix, const
vector<string>& anita_2311102017) {
    cout << "\t";
    for (const auto& city : anita_2311102017) {
        cout << city << "\t";
    }
    cout << endl;

    for (int i = 0; i < matrix.size(); ++i) {
        cout << anita_2311102017[i] << "\t";
        for (int j = 0; j < matrix[i].size(); ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

int main() {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;

    vector<string> anita_2311102017(n);
    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> anita_2311102017[i];
```

```
    }

    vector<vector<int>> adjMatrix(n, vector<int>(n, 0));
    cout << "Silakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i != j) {
                cout << anita_2311102017[i] << " ---> " <<
anita_2311102017[j] << " = ";
                cin >> adjMatrix[i][j];
            }
        }
    }

    printMatrix(adjMatrix, anita_2311102017);

    return 0;
}
```

OUTPUT PROGRAM

```
Silakan masukan jumlah simpul: 3
Silakan masukan nama simpul
Simpul 1: MKS
Simpul 2: DIY
Simpul 3: PWT
Silakan masukkan bobot antar simpul
MKS ---> DIY = 2
MKS ---> PWT = 5
DIY ---> MKS = 2
DIY ---> PWT = 3
PWT ---> MKS = 5
PWT ---> DIY = 3
        MKS     DIY     PWT
MKS     0       2       5
DIY     2       0       3
PWT     5       3       0
```

DESKRIPSI PROGRAM

Program di atas merupakan sebuah program C++ yang memungkinkan pengguna untuk memasukkan jumlah simpul, nama-nama simpul, serta bobot antar simpul dalam sebuah graf. Setelah memasukkan data tersebut, program akan mencetak matriks ketetanggaan yang merepresentasikan graf tersebut, di mana setiap baris dan kolom mewakili simpul-simpul yang dimasukkan, dan nilai dalam matriks menunjukkan bobot koneksi antar simpul-simpul. Program ini menggunakan vektor untuk menyimpan nama-nama simpul dan bobot-bobot koneksi, serta matriks untuk menyimpan informasi ketetanggaan graf..

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

SOURCE CODE

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void printMatrix(vector<vector<int>> &matrix, vector<string>
&anita_2311102017) {
    cout << "\t";
    for (const auto &city : anita_2311102017) {
        cout << city << "\t";
    }
    cout << endl;

    for (int i = 0; i < matrix.size(); ++i) {
        cout << anita_2311102017[i] << "\t";
        for (int j = 0; j < matrix[i].size(); ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}
```

```
void inputTreeData(vector<string> &anita_2311102017,
vector<vector<int>> &adjMatrix) {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;

    anita_2311102017.resize(n);
    adjMatrix.resize(n, vector<int>(n, 0));

    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> anita_2311102017[i];
    }

    cout << "Silakan masukan bobot antar simpul" << endl;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i != j) {
                cout << anita_2311102017[i] << " ---> " <<
anita_2311102017[j] << ": ";
                cin >> adjMatrix[i][j];
            }
        }
    }
}

int main() {
    vector<string> anita_2311102017;
    vector<vector<int>> adjMatrix;
    int choice;

    do {
        cout << "\nMenu:\n";
        cout << "1. Input data tree\n";
        cout << "2. Tampilkan matrix adjacency\n";
        cout << "3. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> choice;
```

```
        switch (choice) {
            case 1:
                inputTreeData(anita_2311102017, adjMatrix);
                break;
            case 2:
                if (anita_2311102017.empty() || adjMatrix.empty()) {
                    cout << "Data tree belum diinputkan.\n";
                } else {
                    printMatrix(adjMatrix, anita_2311102017);
                }
                break;
            case 3:
                cout << "Keluar dari program.\n";
                break;
            default:
                cout << "Pilihan tidak valid. Silakan coba lagi.\n";
        }
    } while (choice != 3);

    return 0;
}
```

OUTPUT PROGRAM

```
Menu:
1. Input data tree
2. Tampilkan matrix adjacency
3. Keluar
Pilihan Anda: 1
Silakan masukan jumlah simpul: 3
Silakan masukan nama simpul
Simpul 1: MKS
Simpul 2: DIY
Simpul 3: PWT
Silakan masukan bobot antar simpul
MKS ---> DIY: 2
MKS ---> PWT: 5
DIY ---> MKS: 2
DIY ---> PWT: 3
PWT ---> MKS: 5
PWT ---> DIY: 3
```

```
Menu:
1. Input data tree
2. Tampilkan matrix adjacency
3. Keluar
Pilihan Anda: 2
        MKS     DIY     PWT
MKS     0       2       5
DIY     2       0       3
PWT     5       3       0

Menu:
1. Input data tree
2. Tampilkan matrix adjacency
3. Keluar
Pilihan Anda: 3
Keluar dari program.
```

DESKRIPSI PROGRAM

Program di atas merupakan sebuah program C++ yang memungkinkan pengguna untuk memasukkan data sebuah pohon atau graf berbentuk pohon, kemudian menampilkan matriks ketetanggaan dari pohon atau graf tersebut. Program memiliki tiga pilihan menu: pertama, untuk memasukkan data pohon; kedua, untuk menampilkan matriks ketetanggaan dari pohon yang telah dimasukkan; dan ketiga, untuk keluar dari program. Program menggunakan vektor untuk menyimpan nama-nama simpul dan bobot-bobot koneksi, serta matriks untuk menyimpan informasi ketetanggaan graf. Pemrosesan data dilakukan dengan menggunakan fungsi-fungsi yang dipisahkan ke dalam fungsi-fungsi terpisah untuk meningkatkan keterbacaan dan modularitas program. Program menggunakan loop do-while untuk mengulang menu hingga pengguna memilih untuk keluar dari program.

BAB IV

KESIMPULAN

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau busur. Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data. Terdapat beberapa jenis graph, antara lain directed graph, undirected graph, dan weight graph. Sementara itu, tree atau pohon adalah struktur data yang terdiri dari node-node yang tertaut secara terurut dalam grafik terhubung, di mana setiap node memiliki paling banyak satu simpul induk dan nol atau lebih simpul anak dengan urutan tertentu. Binary tree merupakan salah satu jenis tree di mana setiap simpul memiliki paling banyak dua simpul anak. Operasi pada tree meliputi pembentukan, pengosongan, pengecekan kekosongan, penyisipan, pencarian, pengubahan, pengambilan isi, penghapusan subtree, serta penelusuran dengan metode pre-order, in-order, dan post-order. Metode penelusuran ini masing-masing memiliki alur yang berbeda namun bertujuan untuk mengunjungi dan memproses seluruh node dalam tree secara teratur.

DAFTAR PUSTAKA

Asisten Praktikum. (2024). *MODUL 9: GRAPH DAN TREE.*

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications.