LAPORAN PRAKTIKUM

MODUL IX

GRAPH DAN TREE

Disusun oleh:
Imelda Fajar Awalina Crisyanti
NIM: 2311102004

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I
TUJUAN PRAKTIKUM

a. Mahasiswa diharapkan mampu memahami graph dan tree

b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree padapemrograman

BAB II
DASAR TEORI

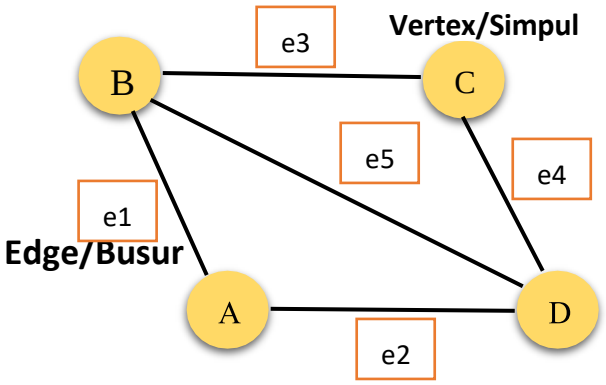
1. Graph

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.

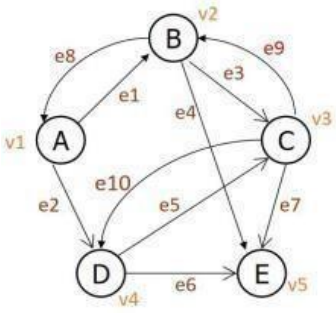
Dapat digambarkan:



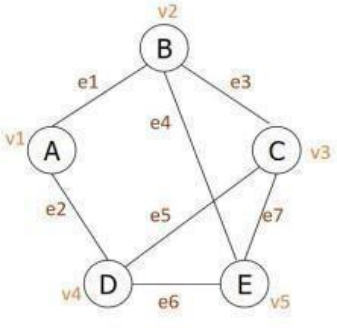
Gambar 1 Contoh Graph

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

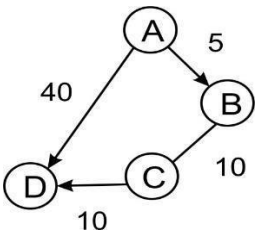
Jenis-jenis Graph



Directed Graph



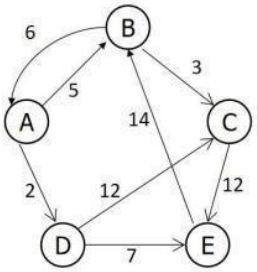
Undirected Graph



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

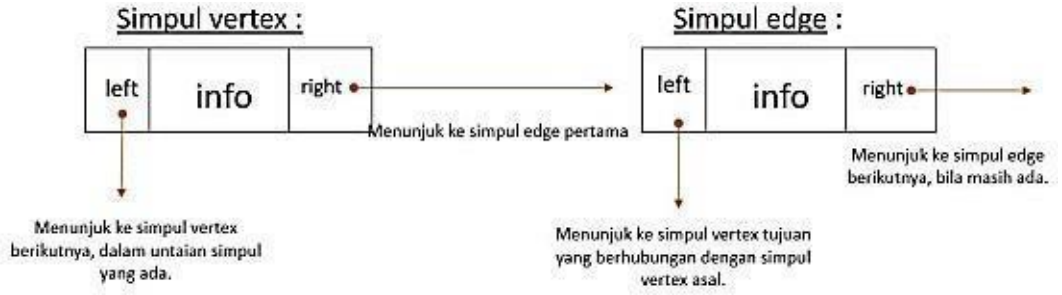


Ke

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

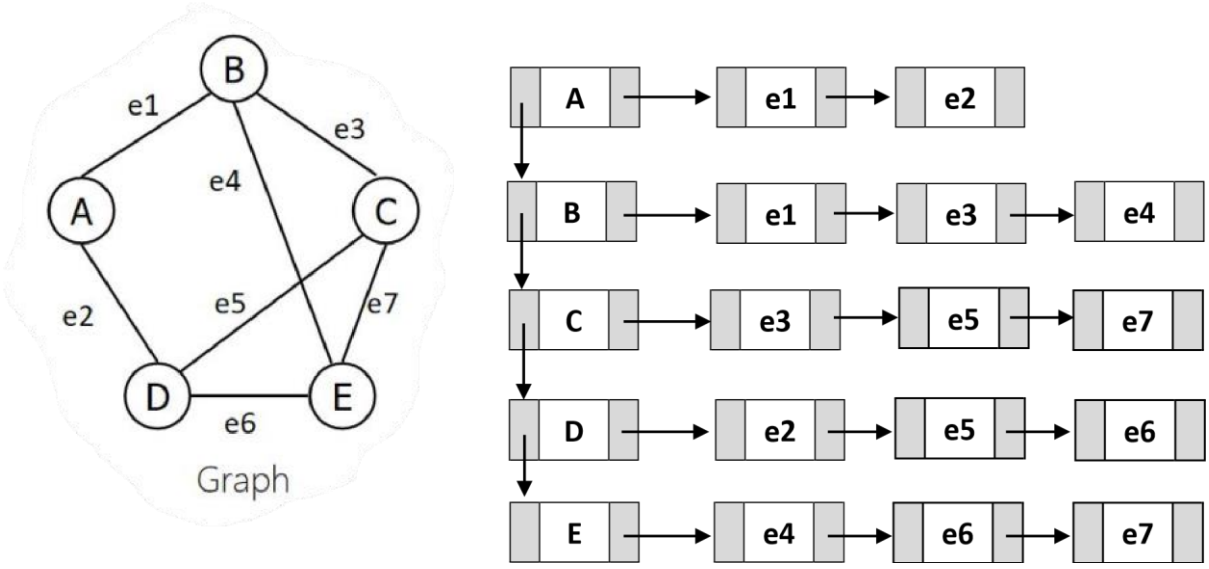
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

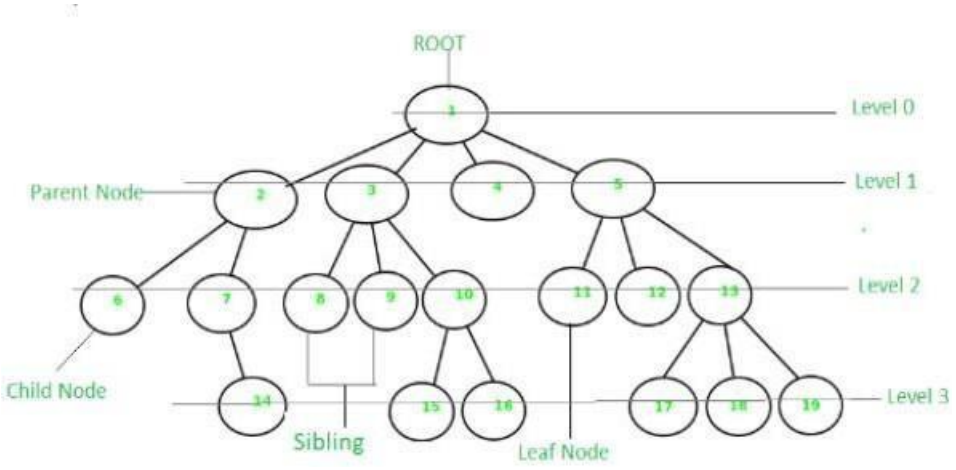
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan,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



Gambar 6 Representasi Graph dengan Linked List

1. Tree atau Pohon

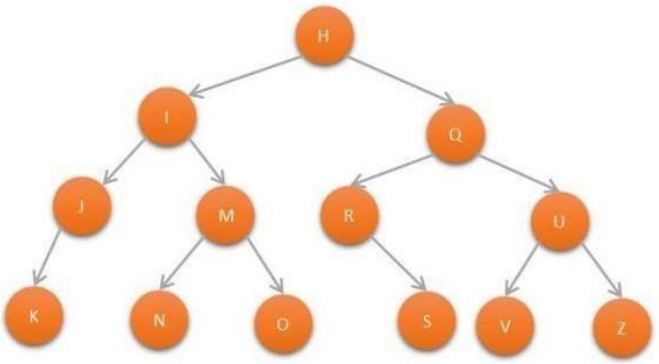
Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. 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

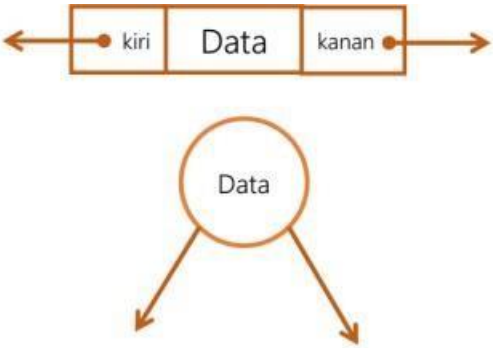
Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



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 ataumenghapus semua node pada binary tree.
c. **isEmpty**: digunakan untuk memeriksa apakah binary tree masih kosong atautidak.
d. **Insert**: digunakan untuk memasukkan sebuah node kedalam tree.
e. **Find**: digunakan untuk mencari root, parent, left child, atau right child dari suatunode 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 seluruhdescendant-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 yakniPre-Order, In-Order, dan Post-Order.

1. Pre-Order

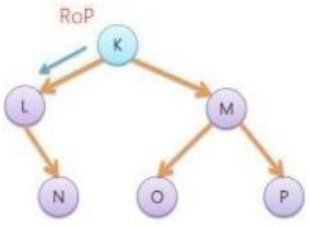
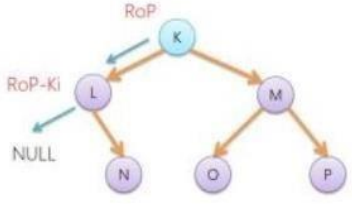
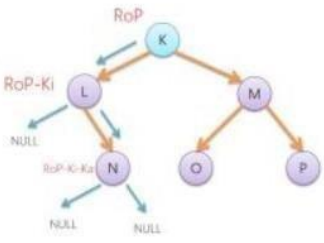
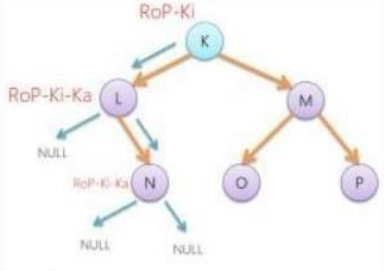
Penelusuran secara pre-order memiliki alur:

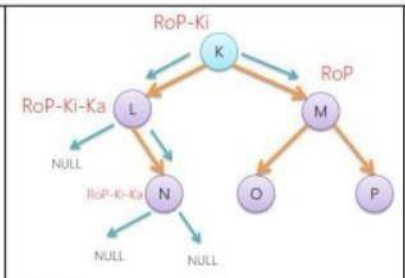
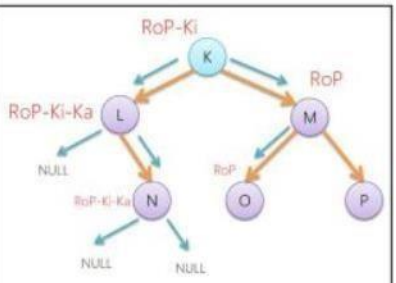
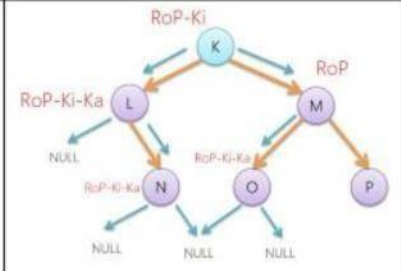
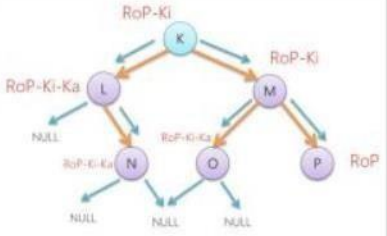
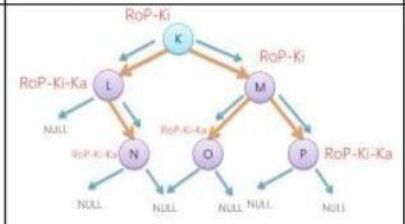
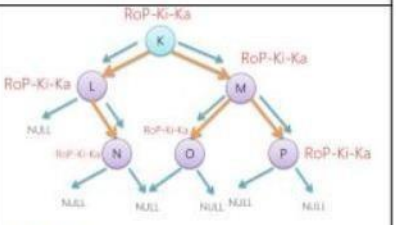
a. Cetak data pada simpul root
b. Secara rekursif mencetak seluruh data pada subpohon kiri
c. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

d. Alur pre-order

No	Proses	No	Proses
1	 K	2	 K - L
3	 K-L-N	4	 K-L-N

5	 K-L-N-M	6	 K-L-N-M-O
7	 K-L-N-M-O	8	 K-L-N-M-O-P
9	 K-L-N-M-O-P	10	 K-L-N-M-O-P

1. In-Order

Penelusuran secara in-order memiliki alur:

e. Secara rekursif mencetak seluruh data pada subpohon kiri

f. Cetak data pada root

g. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

Ro - KiP - Ka

1. Post Order

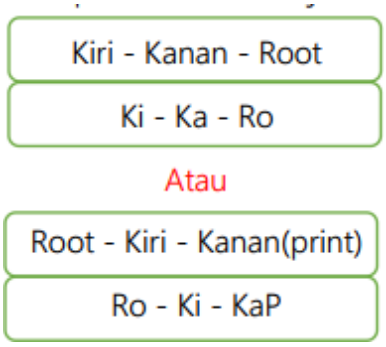
Penelusuran secara in-order memiliki alur:

h. Secara rekursif mencetak seluruh data pada subpohon kiri

i. Secara rekursif mencetak seluruh data pada subpohon kanan

j. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



BAB III

GUIDED

1. **Guided 1**

   Source code

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

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

Screenshoot program

```
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 ini menggunakan array simpul untuk menyimpan nama-nama simpul (node) dalam graf. Ada 7 simpul: "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto", dan "Yogjakarta". Array busur adalah matriks 2D yang merepresentasikan bobot dari busur (edge) antara simpul-simpul tersebut. Jika busur[i][j] memiliki nilai non-nol, ini berarti ada busur dari simpul i ke simpul j dengan bobot tertentu.

Fungsi ini bertujuan untuk menampilkan graf dalam bentuk adjacency list.Fungsi tersebut menggunakan loop untuk melalui setiap simpul (baris dalam matriks).Untuk setiap simpul, fungsi ini akan mencetak nama simpul tersebut diikuti dengan semua simpul yang terhubung langsung beserta bobotnya.

Fungsi main hanya memanggil tampilGraph untuk menampilkan graf yang telah direpresentasikan.

2. Guided 2

Source code

```
#include <iostream>
using namespace std;
// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent; // pointer
};
// pointer global
Pohon *root;
// Inisialisasi
void init()
{
    root = NULL;
}
bool isEmpty()
{
    return root == NULL;
}
Pohon *newPohon(char data)
{
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}
void buatNode(char data)
{
    if (isEmpty())
    {
        root = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi root." <<
endl;
    }
    else
    {
        cout << "\nPohon sudah dibuat" << endl;
```

```
    }
}
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\nNode " << node->data << " sudah ada child kiri!"
                 << endl;
            return NULL;
        }
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child
kiri " << node->data << endl;
                return baru;
        }
    }
}
Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\nNode " << node->data << " sudah ada child kanan!"
<< endl;
            return NULL;
        }
```

```
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child
kanan " << node->data << endl;
                return baru;
        }
    }
}
void update(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah menjadi " <<
data << endl;
        }
    }
}
void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData node : " << node->data << endl;
```

```
        }
    }
}
void find(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->data << endl;
            if (!node->parent)
                cout << "Parent : (tidak punya parent)" << endl;
            else
                cout << "Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node &&
                node->parent->right == node)
                cout << "Sibling : " << node->parent->left->data << endl;
            else if (node->parent != NULL && node->parent->right != node
&& node->parent->left == node)
                cout << "Sibling : " << node->parent->right->data << endl;
            else
                cout << "Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << "Child Kiri : (tidak punya Child kiri)" << endl;
            else
                cout << "Child Kiri : " << node->left->data << endl;
            if (!node->right)
                cout << "Child Kanan : (tidak punya Child kanan)" << endl;
            else
                cout << "Child Kanan : " << node->right->data << endl;
        }
    }
}
// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node)
```

```
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}
// inOrder
void inOrder(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}
// postOrder
void postOrder(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat 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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    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 (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil dihapus."
             << endl;
    }
}
```

```
// Hapus Tree
void clear()
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat 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)
{
    if (isEmpty())
    {
        cout << "\nBuat 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()
{
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}
int main()
{
    init();
    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 << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;
    cout << "InOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;
    cout << "PostOrder :" << endl;
    postOrder(root);
    cout << "\n" << endl;
    characteristic();
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;
    characteristic();
}
```

Screenshoot 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

1. Deklarasi Struktur dan Variabel

- **Struct Pohon:** Mendefinisikan struktur Pohon yang memiliki data karakter, serta pointer ke anak kiri (left), anak kanan (right), dan parent.
- **Pointer Global root:** Menunjukkan akar dari binary tree.

2. Fungsi Inisialisasi dan Utility

- **init():** Menginisialisasi tree dengan mengatur root ke NULL.
- **isEmpty():** Mengecek apakah tree kosong dengan memeriksa apakah root adalah NULL.
- **newPohon(char data):** Membuat node baru dengan nilai data dan mengembalikannya.

3. Fungsi untuk Menambah Node

- **buatNode(char data):** Membuat node akar (root) jika tree masih kosong.
- **insertLeft(char data, Pohon *node):** Menambahkan node baru sebagai anak kiri dari node yang diberikan.

- **insertRight(char data, Pohon *node):** Menambahkan node baru sebagai anak kanan dari node yang diberikan.

4. Fungsi untuk Memodifikasi dan Mengambil Data Node

- **update(char data, Pohon *node):** Mengubah data dari node yang diberikan.
- **retrieve(Pohon *node):** Mengambil dan mencetak data dari node yang diberikan.
- **find(Pohon *node):** Menampilkan informasi detail dari node yang diberikan, termasuk data, parent, sibling, dan child.

5. Fungsi Penelusuran (Traversal)

- **preOrder(Pohon *node):** Melakukan traversal pre-order dan mencetak data node.
- **inOrder(Pohon *node):** Melakukan traversal in-order dan mencetak data node.
- **postOrder(Pohon *node):** Melakukan traversal post-order dan mencetak data node.

6. Fungsi Penghapusan Node

- **deleteTree(Pohon *node):** Menghapus seluruh tree atau subtree dari node yang diberikan.
- **deleteSub(Pohon *node):** Menghapus subtree dari node yang diberikan.
- **clear():** Menghapus seluruh tree.

7. Fungsi Karakteristik Tree

- **size(Pohon *node):** Menghitung jumlah node dalam tree atau subtree dari node yang diberikan.
- **height(Pohon *node):** Menghitung tinggi tree atau subtree dari node yang diberikan.
- **characteristic():** Menampilkan ukuran (size), tinggi (height), dan rata-rata node dari tree.

8. Fungsi main()

- Inisialisasi tree dan menambahkan beberapa node.
- Memodifikasi node tertentu.
- Mengambil dan menampilkan data dari node tertentu.
- Melakukan traversal pre-order, in-order, dan post-order.
- Menampilkan karakteristik tree.
- Menghapus subtree tertentu dan menampilkan kembali tree serta karakteristiknya.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

```cpp
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;

int main() {
    int jumlahSimpul_2311102004;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul_2311102004;

    vector<string> simpul(jumlahSimpul_2311102004);
    for (int i = 0; i < jumlahSimpul_2311102004; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    vector<vector<int>> bobot(jumlahSimpul_2311102004,
vector<int>(jumlahSimpul_2311102004, 0));

    for (int i = 0; i < jumlahSimpul_2311102004; i++) {
        for (int j = 0; j < jumlahSimpul_2311102004; j++) {
            if (i != j) {
                cout << simpul[i] << " --> " << simpul[j] << " = ";
                cin >> bobot[i][j];
            }
        }
    }

    cout << "\n";
    cout << setw(10) << "";
    for (int i = 0; i < jumlahSimpul_2311102004; i++) {
        cout << setw(10) << simpul[i];
    }
    cout << "\n";
```

```
    for (int i = 0; i < jumlahSimpul_2311102004; i++) {
        cout << setw(10) << simpul[i];
        for (int j = 0; j < jumlahSimpul_2311102004; j++) {
            cout << setw(10) << bobot[i][j];
        }
        cout << "\n";
    }

    return 0;
}
```

Screenshoot program

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

                BALI      PALU
      BALI         0         0
      PALU         3         0
```

Deskripsi program

Program di atas adalah sebuah program C++ yang memungkinkan pengguna untuk memasukkan jumlah simpul dalam sebuah graf, serta bobot dari setiap sisi dalam graf tersebut. Program akan meminta pengguna untuk memasukkan jumlah simpul, kemudian nama dari setiap simpul, dan akhirnya memasukkan bobot untuk setiap sisi yang menghubungkan simpul-simpul tersebut.

Program meminta pengguna untuk memasukkan jumlah simpul. Setelah itu, program membuat sebuah vektor simpul yang akan menyimpan nama-nama simpul yang dimasukkan pengguna.Kemudian, program membuat sebuah matriks bobot yang akan menyimpan bobot dari setiap sisi dalam graf. Awalnya, semua bobot diinisialisasi menjadi 0. Program kemudian meminta pengguna untuk memasukkan bobot untuk setiap sisi yang menghubungkan dua simpul yang berbeda. Setelah semua bobot dimasukkan, program mencetak matriks bobot tersebut ke layar dalam format yang terstruktur, dengan menyertakan nama simpul di setiap baris dan kolom. Akhirnya, program selesai dieksekusi.

2. Unguided 2

Source code

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

using namespace std;

void printTree(const vector<string>& simpul, const
vector<vector<int>>& bobot) {
    cout << "\n";
    cout << setw(10) << "";
    for (size_t i = 0; i < simpul.size(); i++) {
        cout << setw(10) << simpul[i];
    }
    cout << "\n";

    for (size_t i = 0; i < simpul.size(); i++) {
        cout << setw(10) << simpul[i];
        for (size_t j = 0; j < simpul.size(); j++) {
            cout << setw(10) << bobot[i][j];
        }
        cout << "\n";
    }
}

int main() {
    int jumlahSimpul_2311102004;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul_2311102004;

    vector<string> simpul(jumlahSimpul_2311102004);
    for (int i = 0; i < jumlahSimpul_2311102004; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    vector<vector<int>> bobot(jumlahSimpul_2311102004,
vector<int>(jumlahSimpul_2311102004, 0));

    for (int i = 0; i < jumlahSimpul_2311102004; i++) {
```

```
        for (int j = 0; j < jumlahSimpul_2311102004; j++) {
            if (i != j) {
                cout << simpul[i] << " --> " << simpul[j] << " = ";
                cin >> bobot[i][j];
            }
        }
    }

    int pilihan;
    do {
        cout << "\nMenu:\n";
        cout << "1. Tampilkan Tree\n";
        cout << "2. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                printTree(simpul, bobot);
                break;
            case 2:
                cout << "Terima kasih!\n";
                break;
            default:
                cout << "Pilihan tidak valid. Silakan coba lagi.\n";
        }
    } while (pilihan != 2);

    return 0;
}
```

Screenshoot program

```
1. Tampilkan Tree
2. Keluar
Pilihan Anda: 1

              BALI      PALU
      BALI       0         0
      PALU       3         0
```

Deskripsi program

Program meminta pengguna untuk memasukkan jumlah simpul dalam tree. Pengguna diminta untuk memasukkan nama dari setiap simpul.Program kemudian meminta pengguna untuk memasukkan bobot dari setiap sisi yang menghubungkan dua simpul yang berbeda. Setelah semua data dimasukkan, program menampilkan sebuah menu kepada pengguna. Menu tersebut memiliki dua pilihan:

Pilihan 1: Tampilkan Tree - Menampilkan nama-nama simpul dan bobot-bobot yang terkait dalam bentuk matriks. Pilihan 2: Keluar - Keluar dari program.

Program akan terus menampilkan menu dan menunggu input pengguna hingga pengguna memilih untuk keluar dari program.

BAB IV

KESIMPULAN

Praktikum GRAPH DAN TREE memberikan pemahaman yang mendalam tentang konsep dasar dan struktur data dari graf dan tree. Konsep-konsep ini merupakan dasar yang penting dalam berbagai aplikasi komputer dan ilmu komputer.

Melalui praktikum ini, peserta memperoleh pemahaman tentang berbagai algoritma yang berkaitan dengan graf dan tree, seperti algoritma traversal, pencarian jalur terpendek, pencarian jalur terpanjang, dan operasi-operasi dasar lainnya.

Penting untuk memahami kompleksitas waktu dan ruang dari berbagai algoritma yang digunakan dalam graf dan tree. Ini membantu dalam memilih algoritma yang tepat untuk berbagai jenis masalah dan memahami kinerja algoritma dalam skenario tertentu.