

**LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA**

**MODUL IX
“GRAPH DAN TREE”**



Disusun Oleh :

Nama : Fitri Kusumaningtyas
NIM : 2311102068
Kelas : IF 11 B

DOSEN:

WAHYU ANDI SAPUTRA, S.Pd. , M.Eng

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

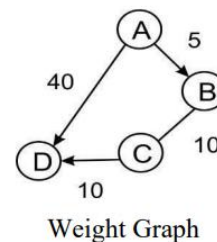
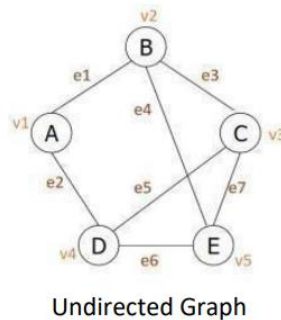
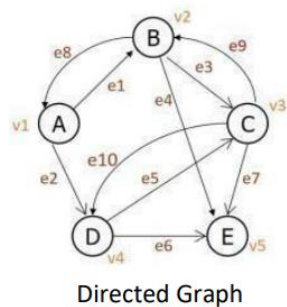
A. DASAR TEORI

1. Graph

Grafik adalah struktur data yang terdiri dari node dan tepi. Node/Vertex menyimpan entitas dan node, sedangkan tepi adalah hubungan antara dua node.

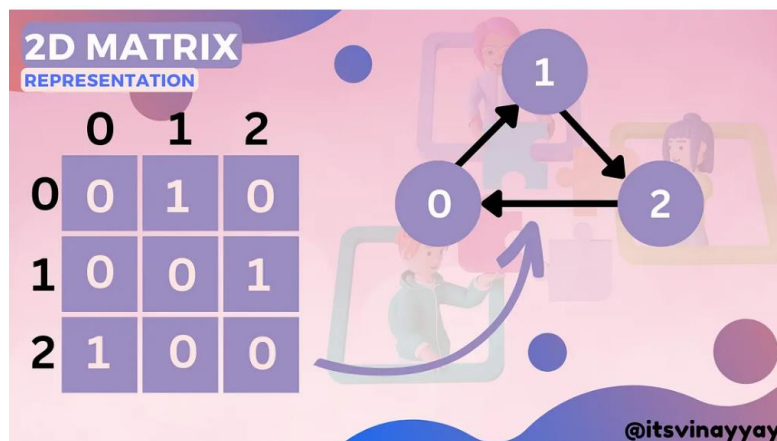
Jenis Grafik :

- ❖ Grafik Berarah dan Tidak Berarah: Ini adalah ketika semua sisi dari sebuah grafik mengarah ke satu node. Jika ada garis sederhana antara dua titik sebagai suatu sisi, graf yang dibuat disebut graf tak berarah.
- ❖ Graf Berbobot: Ini adalah graf dengan bobot setiap sisi, yang merupakan nilai hubungan antara dua node yang dihubungkan oleh sisi mereka.
- ❖ Grafik Siklik dan Asiklik: Dalam grafik berarah, jika ada kemungkinan membuat jalur sedemikian rupa sehingga saat perjalanan Anda berakhir pada node yang sudah dilalui, grafik itu akan disebut Siklik. Sebaliknya berlaku untuk grafik Asiklik.



Representasi Graph:

- ❖ Melalui Matrix 2D

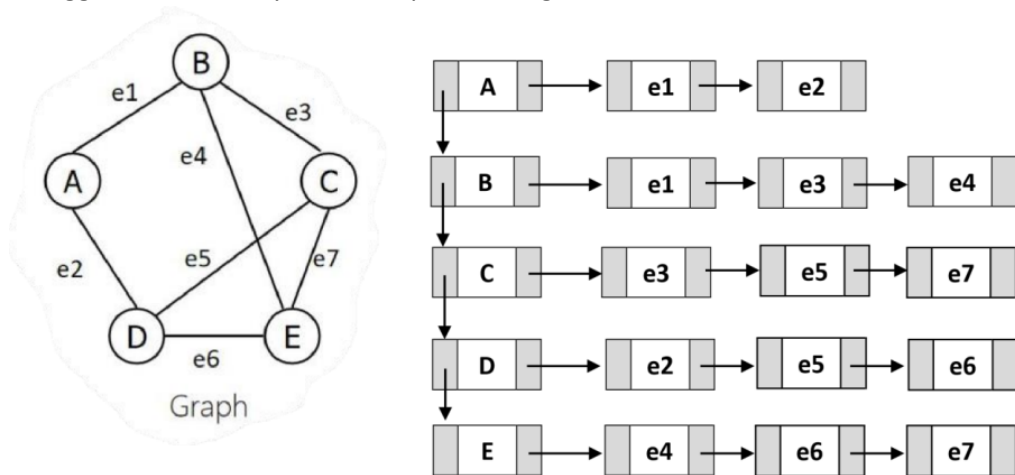


- ❖ Linked list



Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan

menggunakan keduanya dalam representasi graf.



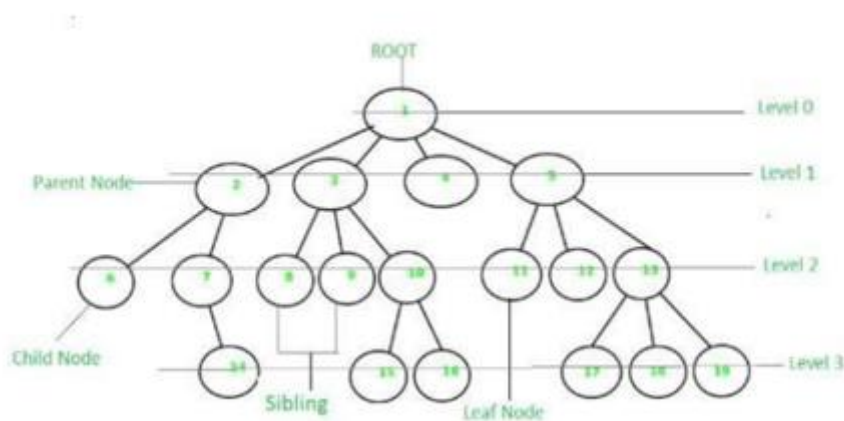
Gambar 6 Representasi Graph dengan Linked List

2. Tree / Pohon

Tree adalah tipe struktur data yang sifatnya non-linier dan berbentuk hierarki. Karena data pada pohon tidak disimpan secara berurutan, tetapi diatur pada beberapa tingkat yang dikenal sebagai struktur hierarkis, pohon dianggap sebagai struktur data non-linier dan berbentuk hierarki. Dalam pohon, hierarki strukturnya mirip dengan hubungan keluarga di mana orang tua dan anak berhubungan satu sama lain. Simpul induk adalah titik tertinggi, dan simpul anak adalah titik yang lebih rendah.

Struktur data pohon terdiri dari kumpulan simpul atau simpul. Setiap simpul pohon menyimpan nilai dan sebuah list rujukan ke simpul lain, yang disebut simpul anak atau simpul anak. Setiap simpul pohon akan dihubungkan oleh sebuah garis hubung, yang disebut tepi. Biasanya dilakukan dengan pointer. Pada pohon, mungkin ada beberapa simpul anak, atau node anak. Namun, hanya satu node dapat mengakses jalan ke child node. Sebuah node atau simpul yang tidak memiliki child node sama sekali disebut leaf node. Struktur data ini adalah cara untuk mengatur dan menyimpan data di komputer agar dapat digunakan dengan lebih baik.

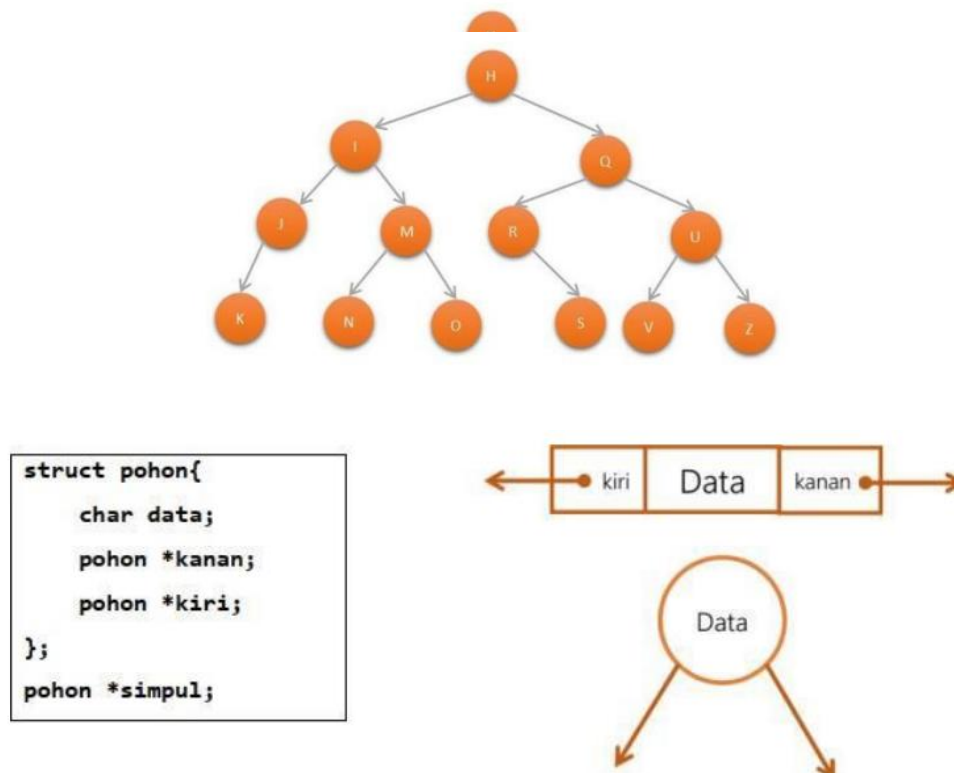
Ilustrasi Struktur pohon sebagai berikut :



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
Root	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Binary tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyarkan memiliki simpul satu level di bawahnya (child) Modul 10 Graph dan Tree tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari 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.



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

- Create:** digunakan untuk membentuk binary tree baru yang masih kosong.
- 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. **Retrieve**: 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 length-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:

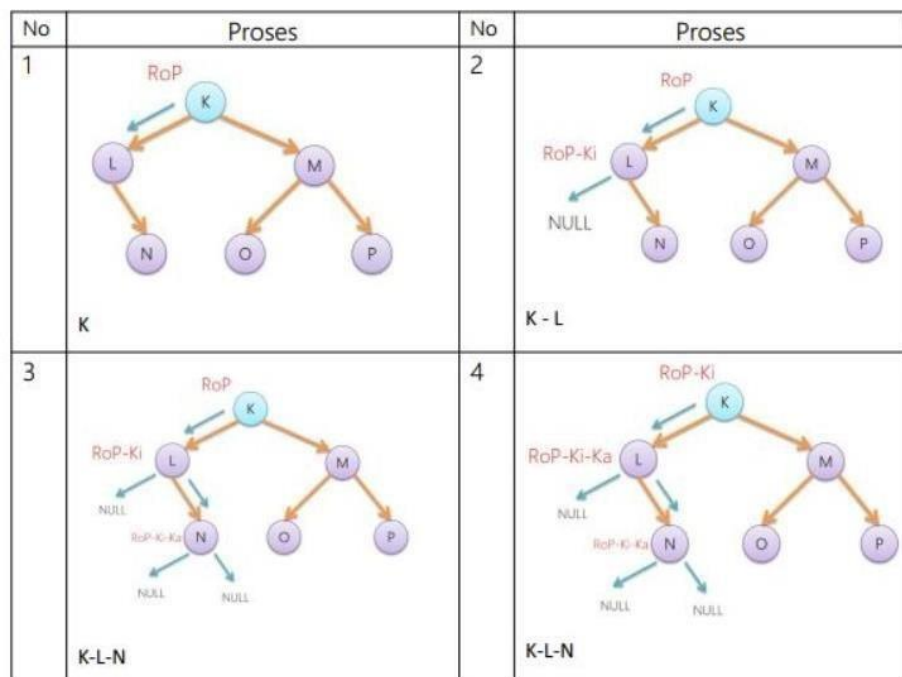
- Cetak data pada simpul root
- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan

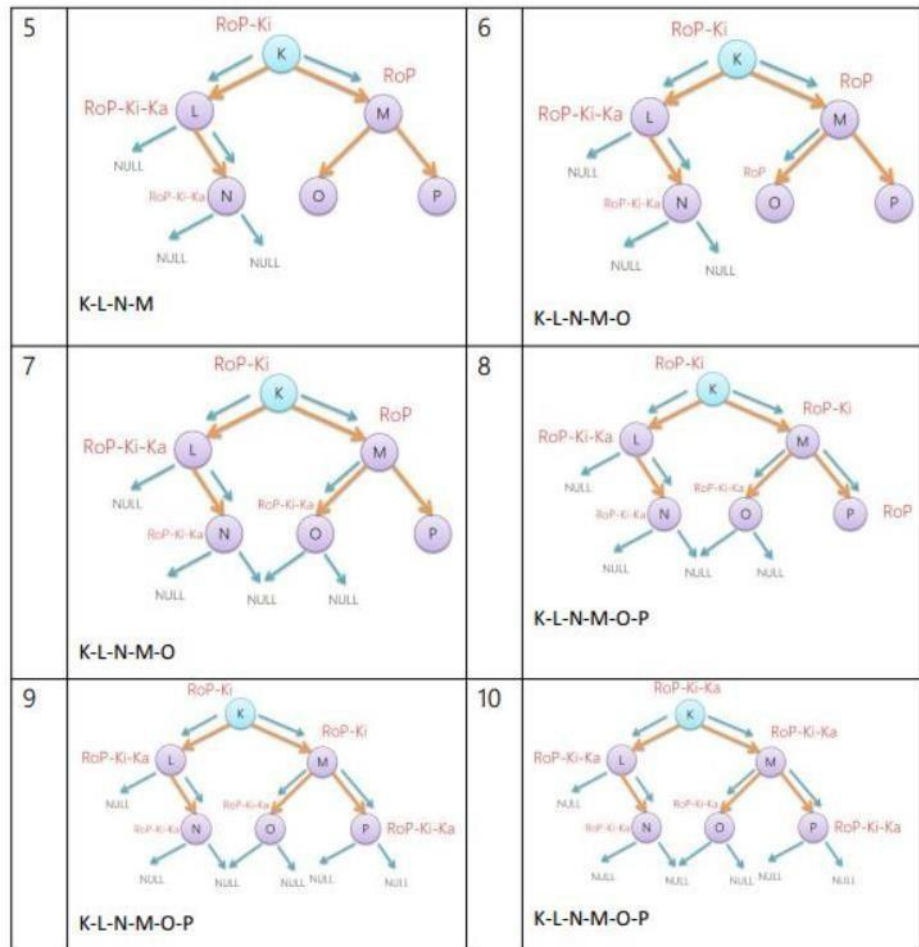
Dapat kita turunkan rumus penelusuran menjadi:

Root (print) - Kiri - Kanan

RoP - Ki - Ka

Alur pre-order





2. In-Order

Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Cetak data pada root
- 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

3. Post Order

Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan
- Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Kanan - Root

Ki - Ka - Ro

Atau

Root - Kiri - Kanan(print)

Ro - Ki - KaP

B. GUIDED

1. Guided I

Source Code

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogjakarta"};

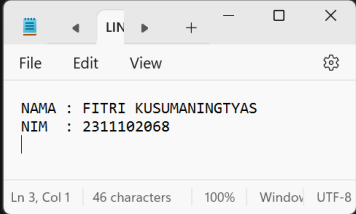
int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

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;
}
```

Ouput Program :

```
PS D:\strukdat> cd "d:\strukdat\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRu
nnerFile }
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:\strukdat>
```



Deskripsi Program :

Berikut adalah deskripsi kode untuk program yang digunakan dalam bahasa C++ untuk menampilkan grafik berbentuk matriks adjacency:

- Deklarasi array simpul yang terdiri dari tujuh string yang mewakili nama-nama kota. deklarasi array busur dua dimensi yang terdiri dari elemen berukuran 7 x 7 yang mewakili bobot atau jarak antar kota. Tidak ada jalur langsung antar kota pada matriks ini, dengan nilai 0.
- Untuk menampilkan grafik berbentuk matriks adjacency, gunakan fungsi Graph(). Fungsi ini mengiterasi setiap baris dan kolom pada matriks busur. Pada baris yang sedang diiterasi, loop menampilkan nama kota, diikuti oleh nama kota yang terkait dengan bobot atau jarak yang sesuai.
- Untuk menampilkan grafik, fungsi main() hanya memanggil fungsi tampilGraph().

2. Guided II

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; // true
    else
        return 0; // false
    }
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat menjadi
root."
            << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada
child kiri!"
                << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;

```

```

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

```

```

    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!!" <<
endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah
menjadi " << data << endl;
        }
    }
}
// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}
// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;

```

```

        cout << " Root : " << root->data << endl;
        if (!node->parent)
            cout << " Parent : (tidak punya parent)" <<
endl;
        else
            cout << " Parent : " << node->parent->data <<
endl;
        if (node->parent != NULL && node->parent->left !=
node &&
            node->parent->right == node)
            cout << " Sibling : " << node->parent->left-
>data << endl;
        else if (node->parent != NULL && node->parent-
>right != node &&
            node->parent->left == node)
            cout << " Sibling : " << node->parent->right-
>data << endl;
        else
            cout << " Sibling : (tidak punya sibling)" <<
endl;
        if (!node->left)
            cout << " Child Kiri : (tidak punya Child
kiri)" << endl;
        else
            cout << " Child Kiri : " << node->left->data <<
endl;
        if (!node->right)
            cout << " Child Kanan : (tidak punya Child
kanan)" << endl;
        else
            cout << " Child Kanan : " << node->right->data
<< endl;
    }
}
// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);

```

```

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

```

```

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

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

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

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

```



```

        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() <<
endl;
}

int main()
{

```

```

    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG,
    *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
        << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
        << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
        << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"
        << endl;
    charateristic();
}

```

Output Program :

```
PS D:\strukdat> cd "d:\strukdat\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

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
PS D:\strukdat> 
```

Deskripsi Program :

Kode di atas merupakan implementasi program untuk pohon binari (binary tree) dalam bahasa C++. Deskripsi kode dalam bahasa Indonesia adalah sebagai berikut:

- Deklarasi struktur node (node) memiliki data, serta pointer ke node induk, node kiri, dan node kanan. Deklarasi pointer root dan baru menunjuk ke node induk dan node baru yang akan dibuat.
- Fungsi `init()` menginisialisasi root menjadi NULL, `isEmpty()` menentukan apakah pohon kosong, dan `buatNode(char data)` membuat node baru dengan data yang diberikan dan menjadikannya root, dan `insertLeft(char data, Pohon *node)` menambahkan node baru sebagai anak kiri dari node yang diberikan.
- Untuk menambahkan node baru sebagai anak kanan dari node yang diberikan, fungsi `insertRight(char data, Pohon *node)`

- fungsi update(char data, Pohon *node) dapat mengubah data node yang diberikan
- fungsi retrieve(Pohon *node) dapat menampilkan data node yang diberikan, parent, sibling, dan anak node.
- Fungsi preOrder(pohon *node) melakukan penelusuran pre-order pada pohon,
- Fungsi inOrder(pohon *node) melakukan penelusuran in-order,
- Fungsi postOrder(pohon *node) melakukan penelusuran post-order,
- Fungsi deleteTree(pohon *node) menghapus node dan semua anaknya dari pohon,
- Fungsi deleteSub(pohon *node) menghapus sub-pohon yang dimulai dari node yang diberikan.
- Untuk menghapus seluruh pohon, gunakan fungsi clear().
- Jumlah (pohon *node) dan tinggi (pohon *node) dihitung oleh fungsi ukuran (pohon *node).
- Karakteristik (pohon *node) menampilkan karakteristik pohon, termasuk jumlah node, tinggi, dan rata-rata jumlah node per tingkat.
- Program membuat pohon dengan root "A" dan menambahkan beberapa node lainnya sebagai anak dan cucu dari root dalam fungsi main(). Kemudian, program mengubah data node, menampilkan data node, melakukan penelusuran, dan menghapus beberapa node.

C. UNGUIDED

1. Unguided 1 Source Code

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

using namespace std;

int main()
{

    int jumlahsimpul_FITRI2311102068;
    cout << "Silakan masukkan jumlah simpul : ";
    cin >> jumlahsimpul_FITRI2311102068;

    vector<string> simpul(jumlahsimpul_FITRI2311102068);
    vector<vector<int>> busur(jumlahsimpul_FITRI2311102068,
vector<int>(jumlahsimpul_FITRI2311102068, 0));

    cout << "Silakan masukkan nama simpul : " << endl;
    for (int i = 0; i < jumlahsimpul_FITRI2311102068; i++)
    {
        cout << "Simpul " << (i + 1) << " : ";
        cin >> simpul[i];
    }

    cout << "Silakan masukkan bobot antar simpul : " << endl;
    for (int i = 0; i < jumlahsimpul_FITRI2311102068; i++)
    {
        for (int j = 0; j < jumlahsimpul_FITRI2311102068;
j++)
        {
            cout << simpul[i] << " --> " << simpul[j] << "
= ";
            cin >> busur[i][j];
        }
    }

    cout << endl;
    cout << setw(7) << " ";
    for (int i = 0; i < jumlahsimpul_FITRI2311102068; i++)
    {
        cout << setw(8) << simpul[i];
    }
}
```

```

        cout << endl;

        for (int i = 0; i < jumlahsimpul_FITRI2311102068; i++)
        {
            cout << setw(7) << simpul[i];
            for (int j = 0; j <
jumlahsimpul_FITRI2311102068; j++)
            {
                cout << setw(8) << busur[i][j];
            }
            cout << endl;
        }
    }
}

```

Output Program :

```

PS D:\strukdat> cd "d:\strukdat\"; if ($?) { g++ prak9Ug1.cpp -o prak9Ug1 }; if ($?) { .\prak9Ug1 }
Silakan masukkan jumlah simpul : 2
Silakan masukkan nama simpul :
Simpul 1 : BALI
Simpul 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:\strukdat>

```

Deskripsi Program :

Program C++ yang digunakan di atas digunakan untuk membuat dan menampilkan grafik matriks adjacency. Deskripsi kode :

- Program meminta pengguna untuk memasukkan jumlah simpul (node) yang akan dibuat. Kemudian, program membuat dua buah vektor: busur untuk menyimpan bobot atau jarak antar simpul dan simpul untuk menyimpan nama-nama simpul.
- Pengguna harus memasukkan nama-nama simpul satu per satu, kemudian program meminta pengguna untuk memasukkan bobot atau jarak antar simpul, dengan nilai untuk setiap pasangan simpul.
- Program menampilkan grafik matriks adjacency dengan nama-nama simpul di baris dan kolom serta bobot atau jarak antar simpul di dalam matriks setelah data dimasukkan.

2. Unguided 2

Source Code

```

#include <iostream>
#include <string>

using namespace std;

struct Node {

```

```

    string data;
    Node* left;
    Node* right;
};

Node* createNode(string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

Node* insertNode(Node* root, string data) {
    if (root == NULL) {
        root = createNode(data);
    } else if (data <= root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}

void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    cout << root->data << " ";
    inorderTraversal(root->right);
}

void displayChild(Node* root, string parent) {
    if (root == NULL) return;
    if (root->data == parent) {
        if (root->left != NULL)
            cout << "Child kiri dari " << parent << ": " <<
root->left->data << endl;
        if (root->right != NULL)
            cout << "Child kanan dari " << parent << ": " <<
root->right->data << endl;
        return;
    }
    displayChild(root->left, parent);
    displayChild(root->right, parent);
}

void displayDescendant(Node* root, string parent) {
    if (root == NULL) return;

```

```

        if (root->data == parent) {
            cout << "Descendant dari " << parent << ": ";
            inorderTraversal(root->left);
            inorderTraversal(root->right);
            cout << endl;
            return;
        }
        displayDescendant(root->left, parent);
        displayDescendant(root->right, parent);
    }

void program_FITRI2311102068() {
    Node* root = NULL;
    int choice;
    string data, parent;

    do {
        cout << "===== " <<
endl;
        cout << "                MENU                " <<
endl;
        cout << "===== " <<
endl;
        cout << "1. Masukan Node" << endl;
        cout << "2. Menampilkan inorder traversal" << endl;
        cout << "3. Menampilkan node child " << endl;
        cout << "4. Menampilkan node descendant " << endl;
        cout << "5. Keluar" << endl;
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan data untuk node baru: ";
                cin >> data;
                root = insertNode(root, data);
                break;
            case 2:
                cout << "Inorder traversal tree: ";
                inorderTraversal(root);
                cout << endl;
                break;
            case 3:
                cout << "Masukkan nama node yang ingin
ditampilkan child-nya: ";
                cin >> parent;
                displayChild(root, parent);
                break;

```



```

        case 4:
            cout << "Masukkan nama node yang ingin
ditampilkan descendant-nya: ";
            cin >> parent;
            displayDescendant(root, parent);
            break;
        case 5:
            cout << "Anda Telah Keluar dari Program, Terima
kasih!\n";
            break;
        default:
            cout << "Pilihan tidak valid!\n";
    }
} while (choice != 5);
}

int main() {
    program_FITRI2311102068();
    return 0;
}

```

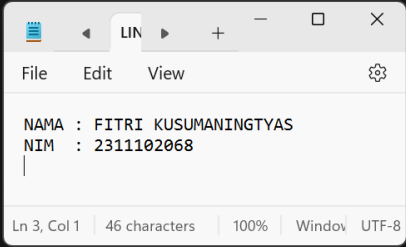
Output Program :

```

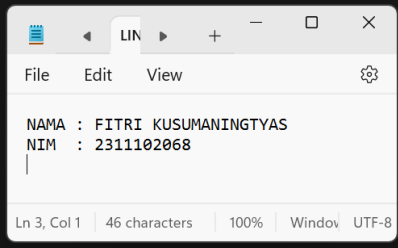
PS D:\strukdat> cd "d:\strukdat\" ; if ($?) { g++ prak9Ug2.cpp -o prak9Ug2 } ; if ($?) { .\prak9Ug2 }
=====
MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: Purwokerto
=====
MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: Tegal
=====
MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: Kebumen

```

```
=====
                        MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: Cilacap
=====
                        MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 1
Masukkan data untuk node baru: Brebes
=====
                        MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 2
Inorder traversal tree: Brebes Cilacap Kebumen Purwokerto Tegal
```



```
=====
                        MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 3
Masukkan nama node yang ingin ditampilkan child-nya: Purwokerto
Child kiri dari Purwokerto: Kebumen
Child kanan dari Purwokerto: Tegal
=====
                        MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 4
Masukkan nama node yang ingin ditampilkan descendant-nya: Cilacap
Descendant dari Cilacap: Brebes
=====
                        MENU
=====
1. Masukan Node
2. Menampilkan inorder traversal
3. Menampilkan node child
4. Menampilkan node descendant
5. Keluar
Pilihan Anda: 5
Anda Telah Keluar dari Program, Terima kasih!
PS D:\strukdat>
```



Deskripsi Program :

Program ini berfungsi sebagai implementasi struktur data pohon dalam bahasa C++. Dengan program ini, pengguna dapat membuat dan mengelola pohon dengan melakukan operasi seperti menambahkan node, menampilkan traversal order, menampilkan anak node, dan menampilkan node turunan. Deskripsi program :

- Program ini menggunakan struktur data pohon, yang terdiri dari node-node dengan data, anak tengah, dan anak kiri.

- Fungsi `createNode` dan `insertNode` membuat node baru dengan data yang diberikan, dan node baru akan ditempatkan di posisi yang sesuai berdasarkan nilai data yang diberikan.
- Dengan menggunakan fungsi `inorderTraversal`, node-node ditampilkan dalam urutan naik
- fungsi `displayChild` menampilkan node anak dari node yang diberikan.
- Fungsi `displayDescendant` digunakan untuk menampilkan node turunan dari node yang diberikan.

Program ini memiliki menu yang memungkinkan pengguna memilih operasi yang ingin dilakukan, seperti menambahkan node, menampilkan traversal urutan, menampilkan anak node, menampilkan node turun, atau keluar dari program. Pengguna dapat memilih operasi yang ingin dilakukan dengan memasukkan nomor yang sesuai. Kemudian, program akan menampilkan pilihan yang dipilih.

D. KESIMPULAN

Mempelajari teori Graph dan Tree di C++ sangat penting bagi programmer karena berfungsi untuk :

1. **Pemahaman Algoritma:** Memahami algoritma penting seperti pencarian, pengurutan, dan optimasi membutuhkan peta dan pohon.
2. **Efisiensi dan Skalabilitas:** Memilih struktur data yang tepat untuk masalah yang dihadapi, sehingga program menjadi lebih efisien dan skalable.

Membangun Aplikasi Kompleks: Ini digunakan untuk berbagai aplikasi kompleks seperti peta jalan, sistem rekomendasi, jaringan sosial, dan analisis data.

3. **Meningkatkan Kemampuan Pemrograman C++:** Meningkatkan keterampilan algoritmik, menguasai struktur data non-linear, dan membangun kebiasaan pemrograman yang baik.
4. **Manfaat Praktis:** Meningkatkan daya saing Anda, mengatasi masalah sehari-hari, dan menemukan peluang baru.

Untuk meningkatkan kemampuan dan menemukan peluang baru dalam karir, programmer harus mempelajari graph dan tree dalam C++.

E. REFERESNSI JURNAL

FIRLIANA, R., & Kasih, P. (2018). Algoritma dan Pemrograman C++.

Michail, D., Kinable, J., Naveh, B., & Sichi, J. V. (2020). JGraphT—A Java library for graph data structures and algorithms. *ACM Transactions on Mathematical Software (TOMS)*, 46(2), 1-29.

Feng, G., Meng, X., & Ammar, K. (2015, October). DISTINGER: A distributed graph data structure for massive dynamic graph processing. In *2015 IEEE International Conference on Big Data (Big Data)* (pp. 1814-1822). IEEE.

Kundu, A., & Bertino, E. (2008). Structural signatures for tree data structures. *Proceedings of the VLDB Endowment*, 1(1), 138-150.

Madhusudan, P., Qiu, X., & Stefanescu, A. (2012). Recursive proofs for inductive tree data-structures. *ACM SIGPLAN Notices*, 47(1), 123-136.