# LAPORAN PRAKTIKUM
# MODUL 9
# GRAPH DAN TREE


**Disusun oleh:**

**Annisa Al Jauhar**

**NIM: 2311102014**

**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

## A. TUJUAN PRAKTIKUM

1. Mahasiswa diharapkan mampu memahami graph dan tree.
2. Mahasiswa diharapkan mampu mengimplematasikan graph dan tree pada pemograman.

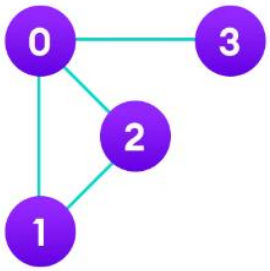
# BAB II
# DASAR TEORI

## B. DASAR TEORI

### 1. **Graph**

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan Simpul pada graph disebut dengan verteks (V), sedangkan sisi yang menghubungkan antar verteks disebut edge (E). Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y.

Sebagai contoh, terdapat graph seperti berikut:




Graph di atas terdiri atas 4 buah verteks dan 4 pasang sisi atau edge. Dengan verteks disimbolkan sebagai V, edge dilambangkan E, dan graph disimbolkan G, ilustrasi di atas dapat ditulis dalam notasi berikut:

V = {0, 1, 2, 3}

E = {(0,1), (0,2), (0,3), (1,2)}

G = {V, E}

Graph banyak dimanfaatkan untuk menyelesaikan masalah dalam kehidupan nyata, dimana masalah tersebut perlu direpresentasikan atau diimajinasikan seperti sebuah jaringan. Contohnya adalah jejaring sosial (seperti Facebook, Instagram, LinkedIn, dkk)

**A. Kelebihan Graph**

Keunggulan dari struktur data graph adalah sbb:

Dengan menggunakan graph kita dapat dengan mudah menemukan jalur terpendek dan tetangga dari node

Graph digunakan untuk mengimplementasikan algoritma seperti DFS dan BFS.

Graph membantu dalam mengatur data.

Karena strukturnya yang non-linier, membantu dalam memahami masalah yang kompleks dan visualisasinya.

**B. Kekurangan Graph**

Adapun kekurangan dari struktur data graph di antaranya:

Graph menggunakan banyak pointer yang bisa rumit untuk ditangani.

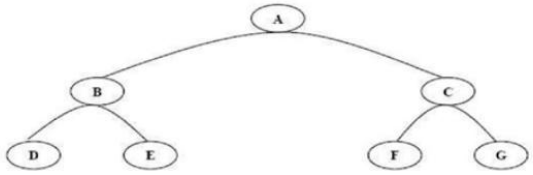
Memiliki kompleksitas memori yang besar.

Jika graph direpresentasikan dengan adjacency matrix maka edge tidak memungkinkan untuk sejajar dan operasi perkalian graph juga sulit dilakukan.

## 2. TREE (POHON)

Tree adalah Kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layakya struktur sebuah pohon. Struktur pohon adalah suatu cara merepresentasikan suatu struktur hirarki (one-to-many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah. Suatu struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one-to-many) dan tidak linier antara elemen-elmennya.

**Ilustrasi alogaritma tree**



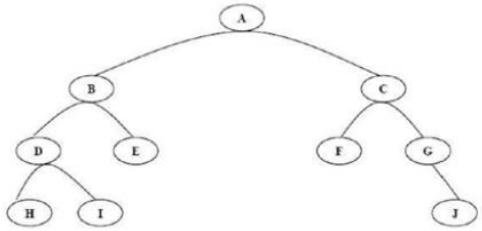

**JENIS-JENIS TREE**

**1. BINARY TREE**

Tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua sub pohon dan kedua subpohon harus terpisah.

Kelebihan struktur Binary Tree :

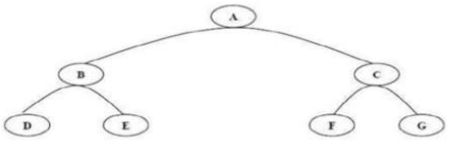
- Mudah dalam penyusunan algoritma sorting
- Searching data relatif cepat
- Fleksibel dalam penambahan dan penghapusan data




Binary Tree

## 2. FULL BINARY TREE

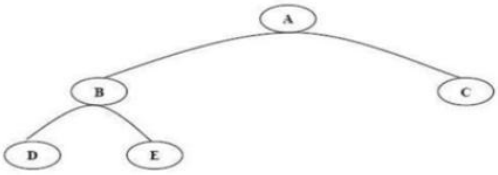
Semua node, kecuali leaf pasti memiliki 2 anak dan tiap subpohon memiliki panjang path yang sama.




Full Binary Tree

## 3. COMPLETE BINARY TREE

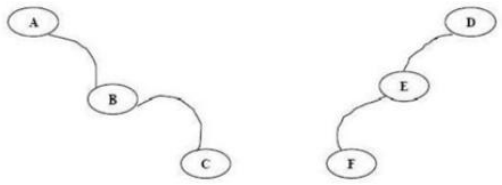
Tree yang mirip dengan full binary tree, tapi tiap subtree boleh memiliki panjang path yang berbeda dan tiap node (kecuali leaf) memiliki 2 anak.




Complete Binary Tree

## 4. SKEWED BINARY TREE

Binary tree yang semua nodenya (kecuali leaf) hanya memiliki satu anak.




Skewed Binary Tree

# BAB III

# GUIDED

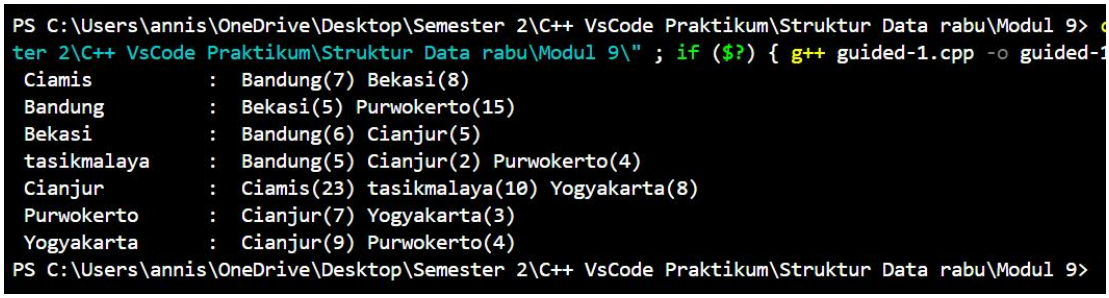
## 1. Guided 1

### Source code

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

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

**Screenshoot program**




## Deskripsi program

Program ini untuk menampilkan struktur graph yang merepresentasikan koneksi antar kota di Indonesia beserta bobotnya. Program menggunakan array simpul untuk menyimpan nama kota dan matriks busur untuk menyimpan bobot koneksi antar kota.

Fungsi tampilGraph menampilkan graph dengan mencetak setiap kota dan koneksi-koneksinya dalam format yang mudah dibaca. Fungsi main hanya memanggil tampilGraph untuk menampilkan graph dan mengakhiri program.

## GUIDED

### 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 << ", ";
        }
    }
}
```

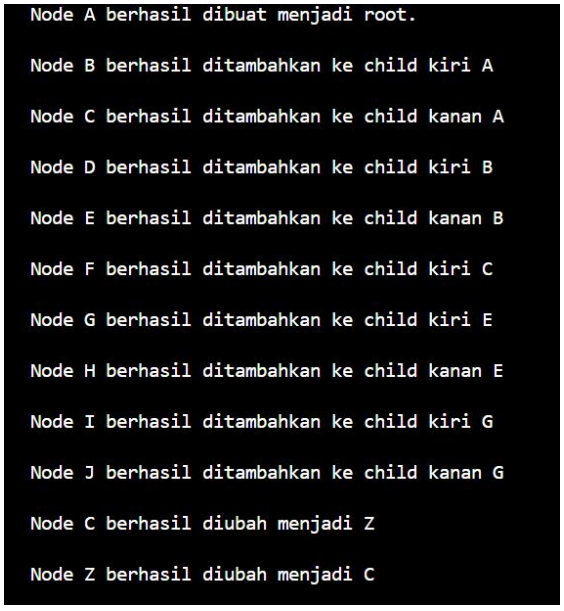
```
// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                if (node->parent->left == node)
                    node->parent->left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



```
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 C:\Users\annis\OneDrive\Desktop\Semester 2\C++ VsCode Praktikum\Struktur Data rabu\Modul 9>
```

## Deskripsi program

Program ini mengimplementasikan pohon biner dalam C++ dengan fungsi untuk membuat, memodifikasi, menelusuri, dan menghapus node. Pohon biner direpresentasikan oleh struktur `Pohon` yang memiliki data karakter dan pointer ke anak kiri, kanan, dan parent, dengan variabel global `root` sebagai root pohon. Fungsi `init()` menginisialisasi root, dan `isEmpty()` mengecek apakah pohon kosong.

Node baru dibuat dengan `newPohon(char data)`, dan fungsi `buatNode(char data)` membuat root jika pohon kosong. Node anak kiri dan kanan ditambahkan dengan `insertLeft` dan `insertRight`. Fungsi `update(char data, Pohon *node)`

mengubah data node, `retrieve(Pohon *node)` menampilkan data node, dan `find(Pohon *node)` menampilkan detail node, termasuk parent, sibling, dan child.

Pohon dapat ditelusuri dengan `preOrder`, `inOrder`, dan `postOrder`. Fungsi `deleteTree(Pohon *node)` menghapus seluruh pohon, `deleteSub(Pohon *node)` menghapus subtree, dan `clear()` menghapus dan menginisialisasi ulang pohon. Fungsi `size(Pohon *node)` menghitung jumlah node, `height(Pohon *node)` menghitung tinggi pohon, dan `characteristic()` menampilkan ukuran, tinggi, dan rata-rata node.

Di `main()`, program menginisialisasi pohon, membuat root, menambahkan node, melakukan operasi update dan retrieve, menampilkan hasil traversal, dan menunjukkan karakteristik pohon. Program juga menghapus subtree dan menampilkan hasil traversal serta karakteristik setelah penghapusan.

# LATIHAN KELAS - UNGUIDED

## 1. Unguided 1

### A. Source code

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

using namespace std;

int main() {
    int num_cities;
    int annisa_al_jauhar_2311102014;
    cout << "Masukkan jumlah kota: ";
    cin >> num_cities;

    vector<string> cities(num_cities);
    cout << "Masukkan nama kota:\n";
    for (int i = 0; i < num_cities; i++) {
        cout << "Kota " << i + 1 << ": ";
        cin >> cities[i];
    }

    vector<vector<int>>                    distances(num_cities,
vector<int>(num_cities, 0));
    cout << "Masukkan jarak antar kota:\n";
    for (int i = 0; i < num_cities; i++) {
        for (int j = 0; j < num_cities; j++) {
            if (i != j) {
                cout << "Jarak dari " << cities[i] << " ke " <<
cities[j] << ": ";
                cin >> distances[i][j];
            }
```

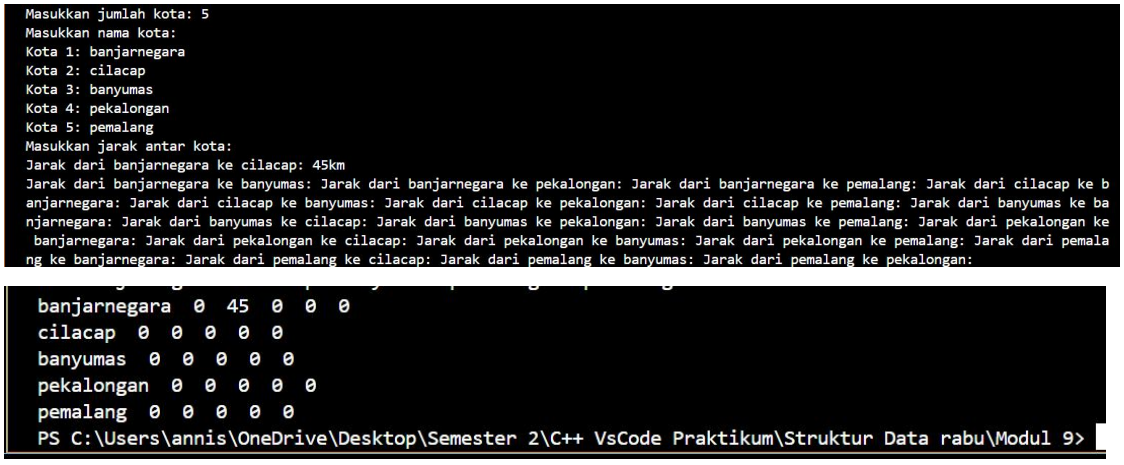
```
        }
    }

    cout << "\n    ";
    for (const auto& city : cities) {
        cout << city << "  ";
    }
    cout << endl;

    // Mencetak baris untuk matriks jarak antar kota
    for (int i = 0; i < num_cities; i++) {
        cout << cities[i] << "  ";
        for (int j = 0; j < num_cities; j++) {
            cout << distances[i][j] << "  ";
        }
        cout << endl;
    }

    return 0;
}
```

## Screenshoot program



## Deskripsi program

Program ini merupakan sebuah aplikasi sederhana untuk mengelola data jarak antara beberapa kota. Program ini meminta pengguna untuk memasukkan jumlah kota dan nama-nama kota tersebut. Kemudian, pengguna diminta untuk memasukkan jarak antar setiap pasangan kota. Data tersebut disimpan dalam bentuk matriks jarak antar kota. Program kemudian mencetak matriks tersebut untuk ditampilkan kepada pengguna.

## 2. Unguided 2

### B. Source code

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

using namespace std;

struct TreeNode {
    string data;
    vector<TreeNode*> children;

    TreeNode(const string& value) : data(value) {}
};

class Tree {
private:
    TreeNode* root;

public:
    Tree() : root(nullptr) {}

    void insertNode(TreeNode* parent, const string& value) {
        TreeNode* newNode = new TreeNode(value);
        if (!root) {
            root = newNode;
            cout << "Node " << value << " berhasil ditambahkan
sebagai root.\n";
        } else {
            parent->children.push_back(newNode);
            cout << "Node " << value << " berhasil ditambahkan
sebagai child dari " << parent->data << ".\n";
        }
```

```
    }

    void displayChildAndDescendant(TreeNode* node) {
        if (!node) return;

        if (!node->children.empty()) {
            cout << "Node " << node->data << " memiliki
child:\n";
            for (TreeNode* child : node->children) {
                cout << child->data << " ";
            }
            cout << endl;
        }

        for (TreeNode* child : node->children) {
            displayChildAndDescendant(child);
        }
    }

    TreeNode* getRoot() const {
        return root;
    }
};

int main() {
    Tree tree;
    TreeNode* parentNode = nullptr;

    int numNodes;
    cout << "Masukkan jumlah node tree: ";
    cin >> numNodes;

    for (int i = 0; i < numNodes; i++) {
        string nodeName;
```

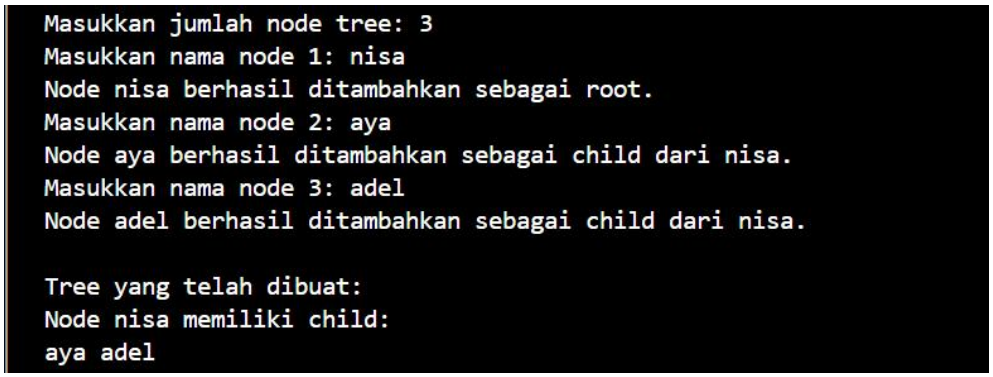
```
        cout << "Masukkan nama node " << i + 1 << ": ";
        cin >> nodeName;

        if (!tree.getRoot()) {
            tree.insertNode(nullptr, nodeName);
            parentNode = tree.getRoot();
        } else {
            tree.insertNode(parentNode, nodeName);
        }
    }

    cout << "\nTree yang telah dibuat:\n";
    tree.displayChildAndDescendant(tree.getRoot());

    return 0;
}
```

## Screenshoot program




## Deskripsi program

Program di atas merupakan implementasi dari struktur data pohon (tree). Program ini memungkinkan pengguna untuk membuat pohon dengan menambahkan node-node ke dalamnya. Setiap node memiliki sebuah string yang menyimpan data, serta sebuah vektor yang berisi pointer ke node-node anaknya. Kelas `TreeNode` mendefinisikan struktur dari setiap node, dengan setiap node

memiliki data dan daftar anak. Kelas `Tree` bertanggung jawab untuk memanajemen pohon, termasuk menyisipkan node baru dan menampilkan node dan keturunannya.

Metode `insertNode` digunakan untuk menambahkan node baru ke dalam pohon. Jika pohon masih kosong, node baru akan menjadi root. Jika tidak, node baru akan ditambahkan sebagai anak dari node yang ditentukan. Metode `displayChildAndDescendant` digunakan untuk menampilkan anak-anak dan keturunan dari suatu node.

Di dalam fungsi `main`, pengguna diminta untuk memasukkan jumlah node yang ingin dibuat. Selanjutnya, pengguna diminta untuk memasukkan nama-nama node tersebut. Setiap node yang dimasukkan akan ditambahkan ke pohon. Setelah semua node ditambahkan, pohon beserta seluruh anak dan keturunannya akan ditampilkan.

# BAB IV

# KESIMPULAN

Kesimpulan dari program ini adalah sebagai berikut:

1. Dengan demikian, dapat disimpulkan bahwa graph G merupakan sebuah pasangan himpunan yang mana di dalamnya akan terdiri dari himpunan tak kosong dan himpunan rusuk yang keduanya kemudian disimbolkan dengan (V dan E).
2. Tree atau pohon adalah struktur data non-linear yang merepresentasikan hubungan hierarkis antar elemen dengan satu elemen sebagai root dan sisanya sebagai simpul. Setiap simpul berisi data dan penunjuk cabang, memiliki tingkat yang dihitung dari akar, derajat yang menyatakan jumlah anak, dan ketinggian yang merupakan tingkat maksimum dikurangi satu. Istilah-istilah terkait termasuk ancestor, predecessor, successor, descendant

## DAFTAR PUSTAKA


1. **https://www.nblognlife.com/2014/12/tree-pada-c-tree-awal.html**
2. **https://www.trivusi.web.id/2022/07/struktur-data-graph.html**