LAPORAN PRAKTIKUM
MODUL IX
GRAPH DAN TREE

Disusun oleh:

Raihan Ramadhan

NIM: 2311102040

Dosen Pengampu:

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

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2024

BAB I

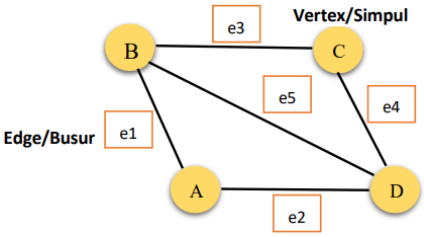
TUJUAN PRAKTIKUM

Praktikum ini bertujuan untuk memastikan bahwa mahasiswa memiliki pemahaman yang baik tentang apa itu graph dan tree, bagaimana cara kerjanya, dan mengapa penting dalam pemrograman. Mahasiswa diharapkan dapat menjelaskan dengan jelas apa graph dan tree dan bagaimana ia bekerja sebagai bagian dari struktur data. Praktikum ini memberikan kesempatan kepada mahasiswa untuk mengimplementasikan graph dan tree dalam bahasa pemrograman c++. Melalui latihan konkret, mahasiswa dapat belajar bagaimana mendefinisikan dan menggunakan graph dan tree dalam konteks nyata. Mahasiswa mungkin akan diberikan situasi atau masalah yang memerlukan penggunaan graph dan tree, dan mereka akan belajar bagaimana cara menguji keefektifan solusi mereka menggunakan graph dan tree.

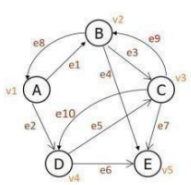
BAB II
DASAR TEORI

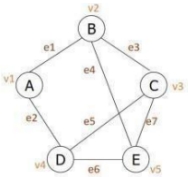
1. Graph

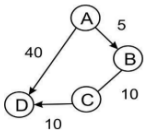
Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai : G = (V, E) Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Dapat digambarkan:



Jenis-jenis Graph

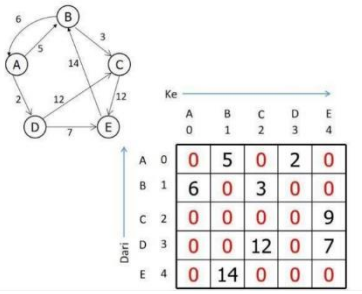

Directed Graph


Undirected Graph



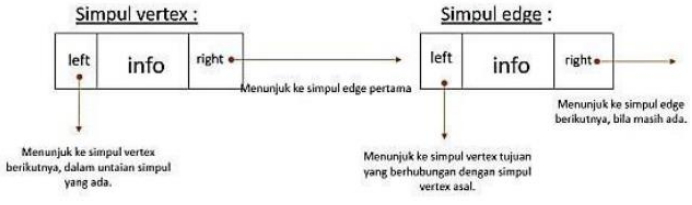
Weight Graph

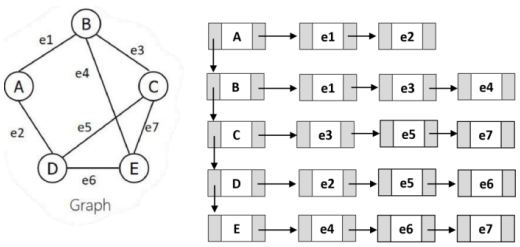
Representasi Graph Representasi dengan Matriks



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.

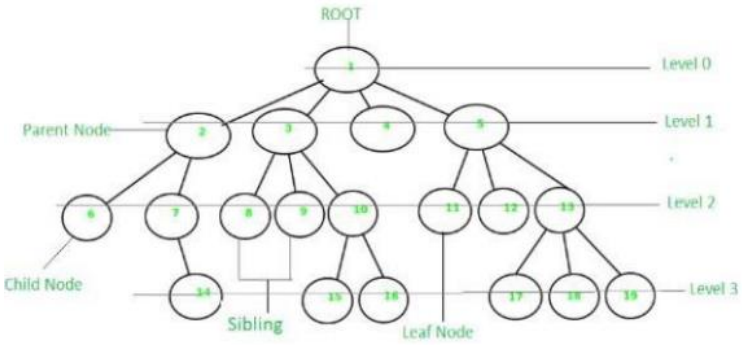
Representasi dengan Linked List





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

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.

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.

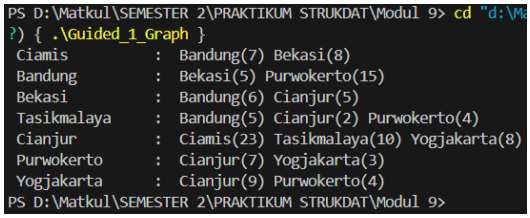
BAB III

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;
}
```

Screenshot Output



Deskripsi Program

Program di atas adalah sebuah implementasi sederhana dari representasi graf berbobot. Graf ini terdiri dari tujuh simpul yang mewakili kota-kota tertentu dan disimpan dalam array string simpul. Matriks busur digunakan untuk menyimpan bobot atau jarak antar simpul, di mana nilai nol menunjukkan tidak adanya busur (atau koneksi) antara dua simpul tersebut. Fungsi tampilGraph() bertanggung jawab untuk menampilkan graf tersebut dengan mencetak setiap simpul dan koneksinya beserta bobotnya. Dalam fungsi main(), program memanggil tampilGraph() untuk menampilkan graf dan kemudian mengakhiri eksekusi.

2. Guided 1

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 Output

```
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:\Matkul\SEMESTER 2\PRAKTIKUM STRUKDAT\Modul 9>
```

Deskripsi Program

Program di atas merupakan implementasi dari struktur data pohon biner .Program ini mendeklarasikan struktur Pohon yang memiliki data, anak kiri, anak kanan, dan parent. Berbagai fungsi disediakan untuk membuat node baru, menambahkan node ke kiri atau kanan, memperbarui data node, mencari node, dan menampilkan karakteristik pohon seperti ukuran dan tinggi. Program ini juga menyediakan fungsi untuk traversal pohon dalam urutan pre-order, in-order, dan post-order, serta fungsi untuk menghapus node atau seluruh pohon. Pada fungsi main(), program mendemonstrasikan penggunaan berbagai fungsi tersebut dengan membuat pohon, menambah node, memperbarui data, melakukan traversal, menampilkan karakteristik, dan menghapus subtree.

BAB IV

UNGUIDED

1. **Unguided 1**
   Source Code

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main() {
    int n_2311102040;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;
    vector<string> kota(int n_2311102040);
    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i int n_2311102040; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> kota[i];
    }
    vector<vector<int>> graf(int n_2311102040, vector<int>( int
n_2311102040));
    cout << "Silakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < int n_2311102040; ++i) {
        for (int j = 0; j < int n_2311102040; ++j) {
            if (i == j) {
                graf[i][j] = 0;
            } else {
                cout << kota[i] << "-->" << kota[j] << " = ";
                cin >> graf[i][j];
            }
        }
    }
    // Output adjacency matrix
    cout << endl << "    ";
    for (int i = 0; i < int n_2311102040; ++i) {
        cout << kota[i] << " ";
    }
    cout << endl;

    for (int i = 0; i < int n_2311102040; ++i) {
        cout << kota[i] << " ";
        for (int j = 0; j < int n_2311102040; ++j) {
            cout << graf[i][j] << " ";
        }
        cout << endl;
    }
 return 0;
}
```

Screenshot Output

```
PS D:\Matkul\SEMESTER 2\PRAKTIKUM STRUKDAT\Modul 9>
nguided_1 }
Silakan masukan jumlah simpul: 2
Silakan masukan nama simpul
Simpul 1: PWT
Simpul 2: PBG
Silakan masukkan bobot antar simpul
PWT-->PBG = 4
PBG-->PWT = 3

    PWT PBG
PWT 0 4
PBG 3 0
PS D:\Matkul\SEMESTER 2\PRAKTIKUM STRUKDAT\Modul 9>
```

Deskripsi Program

Program di atas adalah implementasi sederhana untuk membuat graf berbobot menggunakan matriks ketetanggaan. Pengguna diminta untuk memasukkan jumlah simpul, kemudian memasukkan nama-nama simpul tersebut. Selanjutnya, pengguna diminta untuk memasukkan bobot antara setiap pasangan simpul, kecuali untuk simpul yang sama (di mana bobotnya adalah 0). Setelah semua data dimasukkan, program menampilkan matriks ketetanggaan yang berisi bobot antara simpul-simpul tersebut. Program ini memberikan gambaran visual tentang hubungan dan bobot antar simpul dalam graf yang dimasukkan oleh pengguna.

2. Unguided 2

Source Code

```cpp
#include <iostream>
#include <string>
#include <queue>

using namespace std;

struct Pohon
{
    char data;
    Pohon *kiri, *kanan, *induk;
};

Pohon *akar;

void inisialisasi()
{
    akar = NULL;
}

bool kosong()
{
    return akar == NULL;
}
```

```
Pohon *buatNodeBaru(char data)
{
    Pohon *node = new Pohon();
    node->data = data;
    node->kiri = NULL;
    node->kanan = NULL;
    node->induk = NULL;
    return node;
}

void buatAkar(char data)
{
    if (kosong())
    {
        akar = buatNodeBaru(data);
        cout << "\nNode " << data << " berhasil dibuat sebagai akar." <<
endl;
    }
    else
    {
        cout << "\nPohon sudah ada." << endl;
    }
}

Pohon *tambahKiri(char data, Pohon *node)
{
    if (kosong())
    {
        cout << "\nBuat pohon terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->kiri != NULL)
        {
            cout << "\nNode " << node->data << " sudah memiliki anak
kiri!" << endl;
            return NULL;
        }
        else
        {
            Pohon *nodeBaru = buatNodeBaru(data);
            nodeBaru->induk = node;
            node->kiri = nodeBaru;
            cout << "\nNode " << data << " berhasil ditambahkan sebagai
anak kiri dari " << node->data << endl;
            return nodeBaru;
        }
    }
}
```

```
Pohon *tambahKanan(char data, Pohon *node)
{
    if (kosong())
    {
        cout << "\nBuat pohon terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->kanan != NULL)
        {
            cout << "\nNode " << node->data << " sudah memiliki anak
kanan!" << endl;
            return NULL;
        }
        else
        {
            Pohon *nodeBaru = buatNodeBaru(data);
            nodeBaru->induk = node;
            node->kanan = nodeBaru;
            cout << "\nNode " << data << " berhasil ditambahkan sebagai
anak kanan dari " << node->data << endl;
            return nodeBaru;
        }
    }
}

void preOrder(Pohon *node)
{
    if (node != NULL)
    {
        cout << " " << node->data << ", ";
        preOrder(node->kiri);
        preOrder(node->kanan);
    }
}

void inOrder(Pohon *node)
{
    if (node != NULL)
    {
        inOrder(node->kiri);
        cout << " " << node->data << ", ";
        inOrder(node->kanan);
    }
}
```

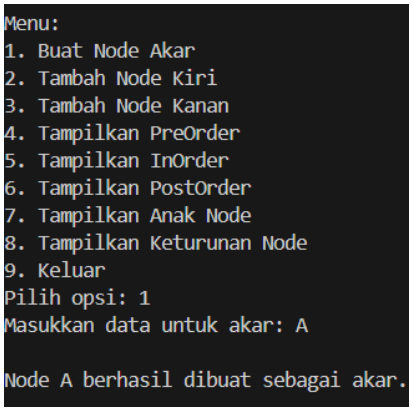
```
void postOrder(Pohon *node)
{
    if (node != NULL)
    {
        postOrder(node->kiri);
        postOrder(node->kanan);
        cout << " " << node->data << ", ";
    }
}

void tampilkanAnak(Pohon *node)
{
    if (node != NULL)
    {
        cout << "Induk: " << node->data << endl;
        if (node->kiri != NULL)
        {
            cout << "Anak Kiri: " << node->kiri->data << endl;
        }
        else
        {
            cout << "Anak Kiri: (tidak punya anak kiri)" << endl;
        }
        if (node->kanan != NULL)
        {
            cout << "Anak Kanan: " << node->kanan->data << endl;
        }
        else
        {
            cout << "Anak Kanan: (tidak punya anak kanan)" << endl;
        }
    }
}

void tampilkanKeturunan(Pohon *node)
{
    if (node != NULL)
    {
        cout << "Keturunan dari " << node->data << ": ";
        queue<Pohon *> q;
        if (node->kiri != NULL)
            q.push(node->kiri);
        if (node->kanan != NULL)
            q.push(node->kanan);
        while (!q.empty())
        {
```

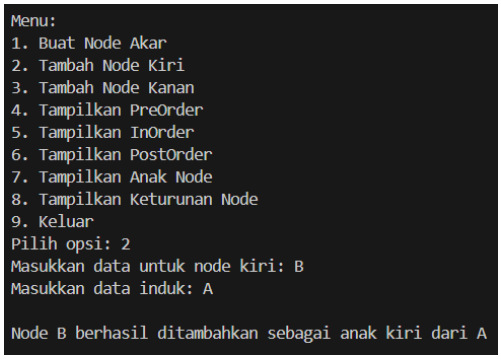
```
            Pohon *sementara = q.front();
            q.pop();
            cout << sementara->data << " ";
            if (sementara->kiri != NULL)
                q.push(sementara->kiri);
            if (sementara->kanan != NULL)
                q.push(sementara->kanan);
        }
        cout << endl;
    }
}

Pohon *cariNode(Pohon *node, char data)
{
    if (node == NULL)
        return NULL;
    if (node->data == data)
        return node;
    Pohon *hasilKiri = cariNode(node->kiri, data);
    if (hasilKiri != NULL)
        return hasilKiri;
    return cariNode(node->kanan, data);
}

int main()
{
    int pilihan;
    char data;
    inisialisasi();

    while (true)
    {
        cout << "\nMenu:\n";
        cout << "1. Buat Node Akar\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
        cout << "4. Tampilkan PreOrder\n";
        cout << "5. Tampilkan InOrder\n";
        cout << "6. Tampilkan PostOrder\n";
        cout << "7. Tampilkan Anak Node\n";
        cout << "8. Tampilkan Keturunan Node\n";
        cout << "9. Keluar\n";
        cout << "Pilih opsi: ";
        cin >> pilihan;
        cin.ignore();
        switch (pilihan)
        {
```

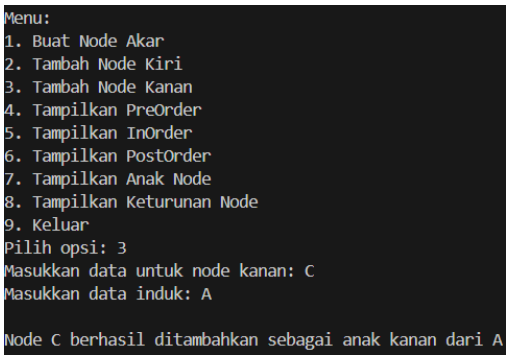
```
        case 1:
            cout << "Masukkan data untuk akar: ";
            cin >> data;
            buatAkar(data);
            break;
        case 2:
        {
            cout << "Masukkan data untuk node kiri: ";
            cin >> data;
            cout << "Masukkan data induk: ";
            char dataInduk;
            cin >> dataInduk;
            Pohon *induk = cariNode(akar, dataInduk);
            if (induk != NULL)
            {
                tambahKiri(data, induk);
            }
            else
            {
                cout << "\nInduk tidak ditemukan!" << endl;
            }
            break;
        }
        case 3:
        {
            cout << "Masukkan data untuk node kanan: ";
            cin >> data;
            cout << "Masukkan data induk: ";
            char dataInduk;
            cin >> dataInduk;
            Pohon *induk = cariNode(akar, dataInduk);
            if (induk != NULL)
            {
                tambahKanan(data, induk);
            }
            else
            {
                cout << "\nInduk tidak ditemukan!" << endl;
            }
            break;
        }
        case 4:
            cout << "\nPreOrder: ";
            preOrder(akar);
            cout << "\n";
            break;
        case 5:
            cout << "\nInOrder: ";
            inOrder(akar);
            cout << "\n";
            break;
```

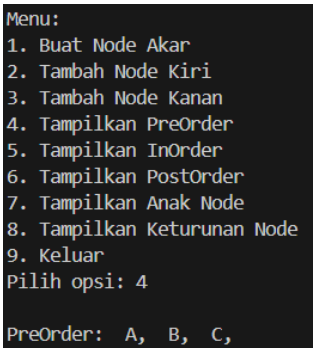
```
        case 6:
            cout << "\nPostOrder: ";
            postOrder(akar);
            cout << "\n";
            break;
        case 7:
            cout << "Masukkan data node untuk menampilkan anak: ";
            cin >> data;
            {
                Pohon *node = cariNode(akar, data);
                if (node != NULL)
                {
                    tampilkanAnak(node);
                }
                else
                {
                    cout << "\nNode tidak ditemukan!" << endl;
                }
            }
            break;
        case 8:
            cout << "Masukkan data node untuk menampilkan keturunan: ";
            cin >> data;
            {
                Pohon *node = cariNode(akar, data);
                if (node != NULL)
                {
                    tampilkanKeturunan(node);
                }
                else
                {
                    cout << "\nNode tidak ditemukan!" << endl;
                }
            }
            break;
        case 9:
            return 0;
        default:
            cout << "Opsi tidak valid! Coba lagi." << endl;
        }
    }

    return 0;
}
```

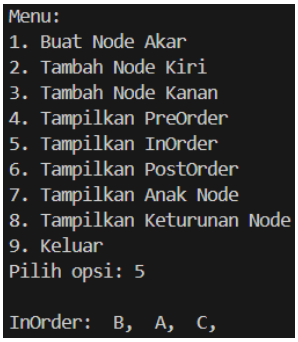
Screenshot Output

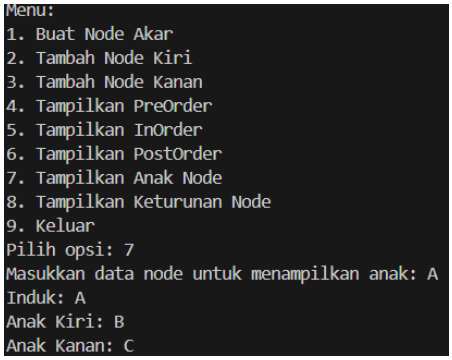


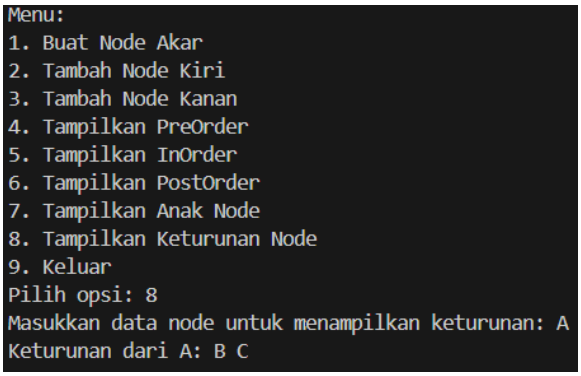












Deskripsi Program

Program di atas adalah implementasi pohon biner (binary tree), yang memungkinkan pengguna untuk membuat dan memanipulasi struktur pohon biner. Pengguna dapat membuat node akar, menambahkan node ke kiri atau kanan dari node yang ada, serta melakukan traversal preorder, inorder, dan postorder. Program ini juga memungkinkan pengguna untuk menampilkan anak-anak (kiri dan kanan) dari node tertentu serta menampilkan semua keturunan dari node tertentu. Terdapat fungsi untuk mencari node berdasarkan data yang dimiliki, memudahkan penambahan node pada posisi yang tepat dalam pohon. Interaksi pengguna dilakukan melalui menu yang memungkinkan pilihan berbagai operasi pada pohon biner.