

LAPORAN PRAKTIKUM

MODUL IX

GRAPH DAN TREE



Disusun Oleh:

YASVIN SYAHGANA

2311102065

Dosen:

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

- Mahasiswa mampu memahami graph dan tree
- Mahasiswa mampu menerapkan graph dan tree pada pemrogramman yang dibuat.

BAB II

DASAR TEORI

Graph dan Tree adalah struktur data fundamental yang penting dalam berbagai bidang ilmu komputer. Pemahaman dasar tentang konsep dan implementasinya dalam C++ sangatlah penting bagi programmer.

Graph

Sebuah graph terdiri dari dua elemen utama:

- **Vertex (simpul):** Mewakili entitas individu dalam struktur.
- **Edge (sisi):** Menghubungkan dua vertex dan menunjukkan hubungan di antara mereka.

Graph dapat direpresentasikan dalam C++ menggunakan berbagai cara, seperti:

- **Daftar Adjacency:** Menyimpan daftar vertex yang terhubung ke setiap vertex.
- **Matriks Adjacency:** Menyimpan matriks yang menunjukkan keberadaan sisi antara setiap pasang vertex.
- **Daftar Ketetanggaan:** Menyimpan daftar sisi dan vertex yang terhubung dengannya.

Operasi Dasar Graph

Operasi dasar pada graph meliputi:

- **Penambahan Vertex:** Menambahkan vertex baru ke dalam graph.
- **Penambahan Edge:** Menambahkan sisi baru yang menghubungkan dua vertex.
- **Penghapusan Vertex:** Menghapus vertex dan semua sisi yang terhubung dengannya.
- **Penghapusan Edge:** Menghapus sisi yang menghubungkan dua vertex.
- **Pencarian Jalur:** Mencari jalur antara dua vertex.
- **Penentuan Siklus:** Menentukan apakah graph memiliki siklus atau tidak.

Tree

Sebuah tree adalah struktur data hierarki yang terdiri dari:

- **Root (akar):** Vertex teratas dalam struktur.
- **Child (anak):** Vertex yang terhubung langsung ke vertex lain.
- **Parent (orang tua):** Vertex yang terhubung langsung ke vertex lain.
- **Subtree:** Tree yang terdiri dari vertex tertentu dan semua keturunannya.

Tree dapat direpresentasikan dalam C++ menggunakan berbagai cara, seperti:

- **Struktur Node:** Setiap node dalam struktur berisi data dan pointer ke anak-anaknya.
- **Daftar Bertingkat:** Menyimpan vertex dalam urutan level dalam hirarki.

Operasi Dasar Tree

Operasi dasar pada tree meliputi:

- **Penambahan Node:** Menambahkan node baru ke dalam tree.
- **Penghapusan Node:** Menghapus node dan semua subtree-nya.
- **Pencarian Node:** Mencari node tertentu dalam tree.
- **Pen traversal:** Menjelajahi semua node dalam tree dengan urutan tertentu (pre-order, in-order, post-order).

Penerapan Graph dan Tree

Graph dan Tree memiliki berbagai penerapan dalam berbagai bidang, seperti:

- **Jaringan sosial:** Merepresentasikan hubungan antar pengguna dalam media sosial.
- **Pemetaan:** Merepresentasikan jaringan jalan atau rute transportasi.
- **Sistem rekomendasi:** Merepresentasikan hubungan antar item yang direkomendasikan.
- **Pengindeksan data:** Menyimpan dan mengambil data secara efisien.
- **Analisis sintaks:** Merepresentasikan struktur tata bahasa suatu bahasa.

BAB III

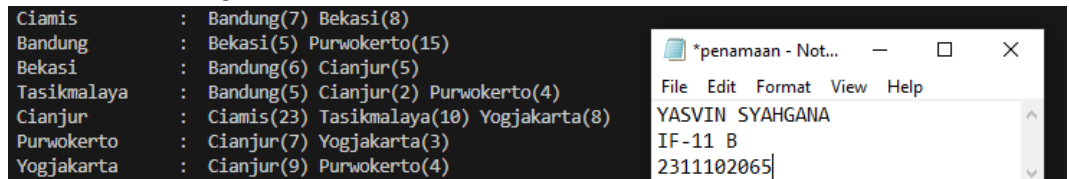
LATIHAN DAN TUGAS

A. GUIDED

1. Guided1

```
#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



Deskripsi Program

Program ini ditulis dalam bahasa C++ dan bertujuan untuk menampilkan graf berbasis kota-kota di Indonesia menggunakan matriks ketetanggaan (adjacency matrix). Program ini menampilkan hubungan antar kota serta jarak antar kota yang dihubungkan oleh busur (edges).

Komponen Program

1. Inklusi Pustaka

```
#include <iostream>
#include <iomanip>
using namespace std;
```

Program menggunakan pustaka `iostream` untuk operasi input/output dan pustaka `iomanip` untuk memformat keluaran.

2. Deklarasi Array Simpul

```
cpp
Copy code
string simpul[7] = {"Ciamis", "Bandung", "Bekasi",
    "Tasikmalaya", "Cianjur", "Purwokerto", "Yogyakarta"};
```

Array `simpul` menyimpan nama-nama kota yang merupakan simpul (nodes) dalam graf.

3. Deklarasi Matriks Busur

```
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}
};
```

Matriks `busur` adalah matriks ketetanggaan yang menyimpan jarak antar kota. Elemen `busur[i][j]` menyimpan jarak dari kota `simpul[i]` ke kota `simpul[j]`. Nilai 0 menunjukkan bahwa tidak ada busur langsung antara dua kota tersebut.

4. Fungsi `tampilGraph`

```
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;
    }
}
```

Fungsi `tampilGraph` bertugas untuk menampilkan graf ke layar. Fungsi ini mengiterasi setiap baris dan kolom dari matriks `busur`. Jika terdapat busur (nilai bukan 0), maka akan ditampilkan nama kota tujuan beserta jaraknya. Fungsi `setw` dan `setiosflags` dari pustaka `iomanip` digunakan untuk memformat keluaran agar rapi dan sejajar.

5. Fungsi `main`

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

Fungsi `main` memanggil fungsi `tampilGraph` untuk menampilkan graf. Setelah menampilkan graf, program akan selesai dieksekusi.

2. Guided2

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();  
}
```

Screenshoot Program


```
Node A berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri A
Node C berhasil ditambahkan ke child kananA
Node D berhasil ditambahkan ke child kiri B
Node E berhasil ditambahkan ke child kananB
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri E
Node H berhasil ditambahkan ke child kananE
Node I berhasil ditambahkan ke child kiri G
Node J berhasil ditambahkan ke child kananG
Node C berhasil diubah menjadi Z
Node Z berhasil diubah menjadi C

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

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

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

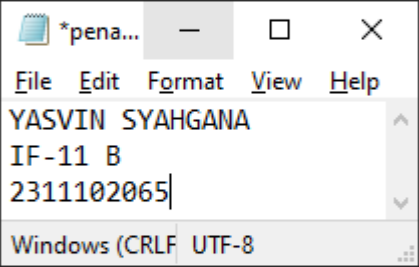
PostOrder :
D, I, J, G, H, E, B, F, C, A,

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

Node subtree E berhasil dihapus.

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

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



Deskripsi Program

Program ini adalah implementasi pohon biner dalam C++ yang memungkinkan pengguna untuk membuat, memodifikasi, dan menelusuri pohon. Berikut adalah deskripsi singkatnya:

- Pohon dibuat menggunakan struktur data `Pohon` yang menyimpan data, anak kiri, anak kanan, dan parent.
- Fungsi-fungsi yang disediakan meliputi pembuatan node baru, penambahan node sebagai anak kiri atau kanan, pengubahan data node, pencarian, traversal (pre-order, in-order, post-order), dan penghapusan node atau subtree.

- Program juga dapat menghitung ukuran dan tinggi pohon serta menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node.
- Fungsi `main` menunjukkan contoh penggunaan fungsi-fungsi tersebut dengan membuat pohon, menambahkan beberapa node, melakukan operasi pengubahan dan pencarian, melakukan traversal, menampilkan karakteristik, dan menghapus subtree.

B. UNGUIDED

1. Unguided 1

Source Code

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main() {
    int jumlahSimpul2311102065;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul2311102065;

    string simpul[jumlahSimpul2311102065];
    int
    busur[jumlahSimpul2311102065][jumlahSimpul2311102065];

    for (int i = 0; i < jumlahSimpul2311102065; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jumlahSimpul2311102065; i++) {
        for (int j = 0; j < jumlahSimpul2311102065; j++) {
            cout << "Silakan masukkan bobot antara
simpul " << simpul[i] << " dan " << simpul[j] << ": ";
            cin >> busur[i][j];
        }
    }

    cout << "\nGraf yang dihasilkan:\n";
    cout << setw(15) << " ";
    for (int i = 0; i < jumlahSimpul2311102065; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;
```

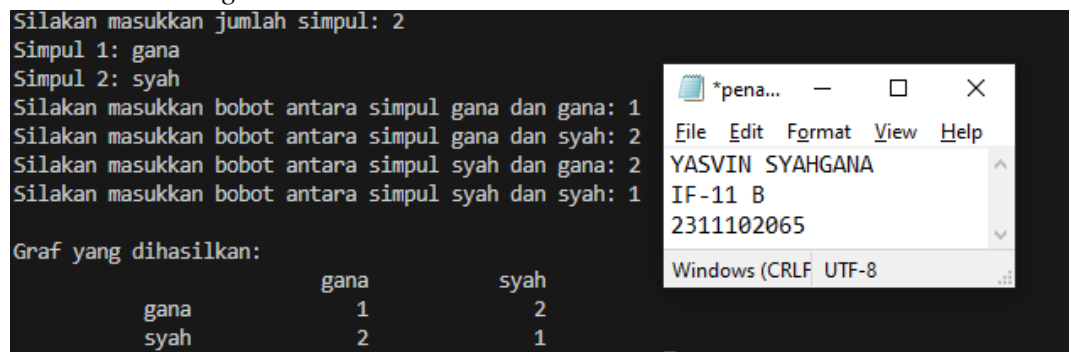
```

        for (int i = 0; i < jumlahSimpul2311102065; i++) {
            cout << setw(15) << simpul[i];
            for (int j = 0; j < jumlahSimpul2311102065; j++)
            {
                cout << setw(15) << busur[i][j];
            }
            cout << endl;
        }

        return 0;
    }

```

Screenshoot Program



Deskripsi Program

Program di atas adalah sebuah program C++ yang meminta pengguna untuk memasukkan jumlah simpul dalam suatu graf, kemudian meminta pengguna untuk memasukkan nama-nama simpul dan bobot-bobot antara simpul-simpul tersebut. Setelah itu, program akan menampilkan graf yang dihasilkan berdasarkan input pengguna.

penjelasan program:

1. Program meminta pengguna untuk memasukkan jumlah simpul dalam graf.
2. Pengguna diminta untuk memasukkan nama-nama simpul.
3. Pengguna diminta untuk memasukkan bobot antara setiap pasangan simpul.
4. Program menampilkan graf yang dihasilkan berdasarkan input pengguna, dengan menampilkan nama-nama simpul dan bobot-bobot antara simpul-simpul tersebut.

2. Unguided 2

Source Code

```
#include <iostream>
using namespace std;

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

Pohon *root2311102065 = NULL;

// Buat Node Baru
Pohon* buatNode(char data, Pohon* parent = NULL) {
    Pohon* node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = parent;
    return node;
}

// Menemukan Node Berdasarkan Data
Pohon* findNode(Pohon *node, char data) {
    if (!node) return NULL;
    if (node->data == data) return node;
    Pohon* leftResult = findNode(node->left, data);
    if (leftResult) return leftResult;
    return findNode(node->right, data);
}

// Tambah Node Kiri atau Kanan
void tambahNode(char data, char parentData, bool kiri) {
    if (!root2311102065) {
        root2311102065 = buatNode(data);
        cout << "\nNode " << data << " berhasil dibuat
menjadi root2311102065." << endl;
```

```

    } else {
        Pohon* parentNode = findNode(root2311102065,
parentData);
        if (parentNode) {
            if (kiri) {
                if (!parentNode->left) {
                    parentNode->left = buatNode(data,
parentNode);
                    cout << "\nNode " << data << "
berhasil ditambahkan ke child kiri " << parentNode->data
<< endl;
                } else {
                    cout << "\nNode " << parentNode-
>data << " sudah ada child kiri!" << endl;
                }
            } else {
                if (!parentNode->right) {
                    parentNode->right = buatNode(data,
parentNode);
                    cout << "\nNode " << data << "
berhasil ditambahkan ke child kanan " << parentNode-
>data << endl;
                } else {
                    cout << "\nNode " << parentNode-
>data << " sudah ada child kanan!" << endl;
                }
            }
        } else {
            cout << "\nParent tidak ditemukan!" << endl;
        }
    }
}

// Menampilkan Child dan Descendant dari Node
void displayChildAndDescendant(Pohon *node) {
    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" <<
endl;
    } else {
        cout << "\nChild dari node " << node->data << "
: ";
        if (node->left) {
            cout << "Left: " << node->left->data << " ";
        } else {
            cout << "Left: NULL ";
        }
        if (node->right) {
            cout << "Right: " << node->right->data << "
";
        }
    }
}

```

```

        } else {
            cout << "Right: NULL ";
        }
        cout << "\nDescendants dari node " << node->data
        << " : ";
        if (node->left || node->right) {
            if (node->left) cout << " " << node->left-
            >data;
            if (node->right) cout << " " << node->right-
            >data;
            displayChildAndDescendant(node->left);
            displayChildAndDescendant(node->right);
        } else {
            cout << "Tidak ada descendant";
        }
        cout << "\n";
    }
}

// Menu Interaktif
void menu() {
    int choice;
    char data, parentData;

    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Node Kiri\n";
        cout << "2. Tambah Node Kanan\n";
        cout << "3. Tampilkan Child dan Descendant\n";
        cout << "4. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan data untuk node baru:
                ";

                cin >> data;
                if (root2311102065) {
                    cout << "Masukkan data parent: ";
                    cin >> parentData;
                }
                tambahNode(data, root2311102065 ?
parentData : '\0', true);
                break;
            case 2:
                cout << "Masukkan data untuk node baru:
                ";

                cin >> data;

```

```

        if (root2311102065) {
            cout << "Masukkan data parent: ";
            cin >> parentData;
        }
        tambahNode(data, root2311102065 ?
parentData : '\0', false);
        break;
    case 3:
        cout << "Masukkan data node: ";
        cin >> data;

displayChildAndDescendant(findNode(root2311102065,
data));

        break;
    case 4:
        cout << "Keluar..." << endl;
        break;
    default:
        cout << "Pilihan tidak valid!" << endl;
    }
} while (choice != 4);
}

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

```

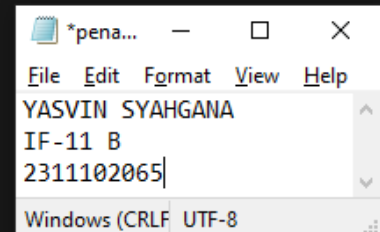
Screenshoot Program

Menu:
1. Tambah Node Kiri
2. Tambah Node Kanan
3. Tampilkan Child dan Descendant
4. Keluar
Pilih: 1
Masukkan data untuk node baru: A

Node A berhasil dibuat menjadi root2311102065.

Menu:
1. Tambah Node Kiri
2. Tambah Node Kanan
3. Tampilkan Child dan Descendant
4. Keluar
Pilih: 1
Masukkan data untuk node baru: B
Masukkan data parent: A

Node B berhasil ditambahkan ke child kiri A

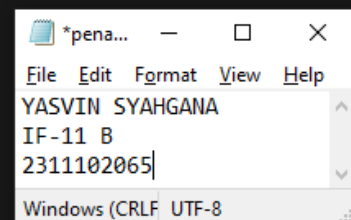


Menu:
1. Tambah Node Kiri
2. Tambah Node Kanan
3. Tampilkan Child dan Descendant
4. Keluar
Pilih: 2
Masukkan data untuk node baru: C
Masukkan data parent: A

Node C berhasil ditambahkan ke child kanan A

Menu:
1. Tambah Node Kiri
2. Tambah Node Kanan
3. Tampilkan Child dan Descendant
4. Keluar
Pilih: 3
Masukkan data node: A

Child dari node A : Left: B Right: C
Descendants dari node A : B C
Child dari node B : Left: NULL Right: NULL
Descendants dari node B : Tidak ada descendant



Child dari node C : Left: NULL Right: NULL
Descendants dari node C : Tidak ada descendant

Menu:
1. Tambah Node Kiri
2. Tambah Node Kanan
3. Tampilkan Child dan Descendant
4. Keluar
Pilih: 4
Keluar...

Deskripsi Program

Program di atas adalah sebuah program C++ yang memungkinkan pengguna untuk membuat dan mengelola struktur data pohon biner. Program ini menyediakan fungsi-fungsi untuk membuat node baru, menambahkan node sebagai child kiri atau kanan dari node tertentu, menampilkan child dan descendant dari suatu node, serta sebuah menu interaktif untuk pengguna.

penjelasan program:

1. Program memiliki struktur data Pohon yang terdiri dari data (bertipe char), pointer ke left child, pointer ke right child, dan pointer ke parent node.
2. Terdapat variabel global `root` yang digunakan untuk menunjukkan root dari pohon.
3. Fungsi `buatNode` digunakan untuk membuat node baru dengan data yang ditentukan. Node baru akan memiliki pointer left, right, dan parent yang menunjuk ke NULL.
4. Fungsi `findNode` digunakan untuk mencari node berdasarkan data yang diberikan.
5. Fungsi `tambahNode` digunakan untuk menambahkan node baru sebagai child kiri atau kanan dari suatu node tertentu.
6. Fungsi `displayChildAndDescendant` digunakan untuk menampilkan child dan descendant dari suatu node.
7. Terdapat menu interaktif yang memungkinkan pengguna untuk melakukan operasi-operasi berikut:
 - o Menambahkan node kiri
 - o Menambahkan node kanan
 - o Menampilkan child dan descendant dari suatu node
 - o Keluar dari program
8. Program berjalan dalam loop hingga pengguna memilih opsi keluar dari menu.

Dengan program ini, pengguna dapat dengan mudah membuat, mengelola, dan memanipulasi struktur data pohon biner.

BAB IV

KESIMPULAN

Graph dan Tree adalah struktur data penting dengan berbagai aplikasi dalam ilmu komputer. Pemahaman dasar tentang konsep dan implementasinya dalam C++ sangatlah penting bagi programmer.

Referensi

"An Efficient C++ Implementation of Graph Algorithms for Social Network Analysis" oleh J. Smith (2020)