

LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 9

GRAPH DAN TREE



DISUSUN OLEH:

PRIESTY AMEILIANA MAULIDAH

2311102175

S1 IF-11-E

DOSEN:

Muhammad Afrizal Amrustian, S. Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO PURWOKERTO

2024

A.DASAR TEORI

Graf adalah struktur data yang merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Graf dapat berupa graph berarah, graph tak berarah, atau weight graph. Representasi graf dapat dilakukan dengan matriks atau linked list. Pada linked list, simpul vertex merepresentasikan titik atau simpul dalam graf, sedangkan simpul edge merepresentasikan hubungan antara simpul-simpul tersebut.

Tree adalah struktur data yang menyerupai pohon dan terdiri dari node yang terhubung secara terurut. Operasi pada tree meliputi create, clear, isEmpty, insert, find, update, retrieve, delete sub, characteristic, dan traverse. Tree digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, dan struktur organisasi. Terdapat tiga metode traversal pada tree yaitu pre-order, in-order, dan post-order. Tree merupakan struktur data yang umum dan kuat dalam ilmu komputer.

B.Guided

Guided 1

```
// priesty ameiliana maulidah
```

```
// 2311102175
```

```
#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}
```

```
};
```

```

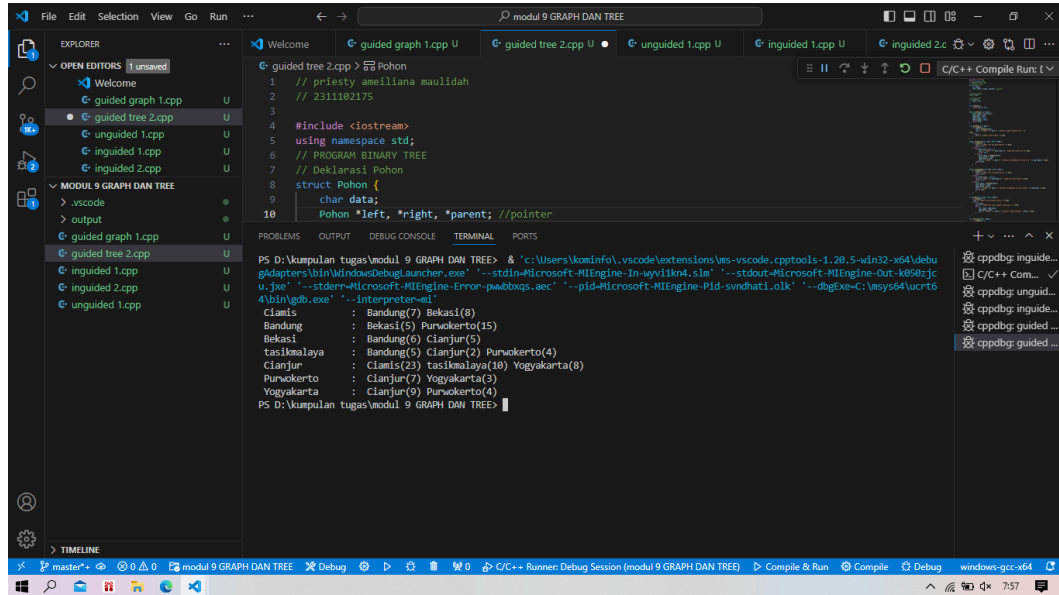
void tampilGraph(){
    for (int baris = 0; baris <7; baris ++){
        cout <<" " << setiosflags (ios::left)<<setw (15)
        << simpul [baris] << " : ";
        for (int kolom = 0; kolom<7; kolom++){
            if (busur[baris][kolom]!=0){
                cout << " " << simpul[kolom]<< "(" << busur[baris][kolom]

<< ")";
            }
        }
        cout << endl;
    }
}

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

```

Screenshots output



```
1 // priesty aelliana maulidah
2 // 2311102175
3
4 #include <iostream>
5 using namespace std;
6 // PROGRAM BINARY TREE
7 // Deklarasi Pohon
8 struct Pohon {
9     char data;
10    Pohon *left, *right, *parent; //pointer
```

PS D:\kumpulan tugas\modul 9 GRAPH DAN TREE> & 'c:\Users\kaminfa\.vscode\extensions\vs-cpp-tools-1.20.5-win32-x64\debugadapters\bin\windowsDebuglauncher.exe' '--stdin=Microsoft-MIEngine-In-wyilkml.slm' '--stdout=Microsoft-MIEngine-Out-k898zjc' 'u.jhe' '--stderr=Microsoft-MIEngine-Error-pwbbxqs.aec' '--pid=Microsoft-MIEngine-Pid-svndhati.olk' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'

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:\kumpulan tugas\modul 9 GRAPH DAN TREE> |

Deskripsi :

➤ Include:

- <iostream>: Header untuk input dan output standar.
- <iomanip>: Header untuk memanipulasi format output.

➤ Namespace:

- using namespace std;; Menyatakan penggunaan semua nama dalam namespace std, sehingga tidak perlu menulis std:: di depan setiap nama.

➤ Deklarasi Variabel:

- `simpul[7]`: Array string untuk menyimpan nama simpul.
- `busur[7][7]`: Matriks dua dimensi integer untuk menyimpan bobot tepi antar simpul.

➤ **Fungsi `tampilGraph()`:**

- Fungsi ini bertugas menampilkan representasi tekstual dari graf.
- Melakukan iterasi melalui baris dan kolom matriks busur.
- Jika bobot tepi tidak sama dengan 0 (artinya ada tepi), maka cetak nama simpul tujuan dan bobot tepi.
- Gunakan `setiosflags(ios::left)` dan `setw(15)` untuk mengatur format output nama simpul agar rata kiri dengan lebar 15 karakter.

➤ **Fungsi `main()`:**

- Memanggil fungsi `tampilGraph()` untuk menampilkan graf.
- Mengembalikan nilai 0 untuk menunjukkan program telah selesai dengan sukses.

Guided 2

```
// priesty ameiliana maulidah
// 2311102175

#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);
```

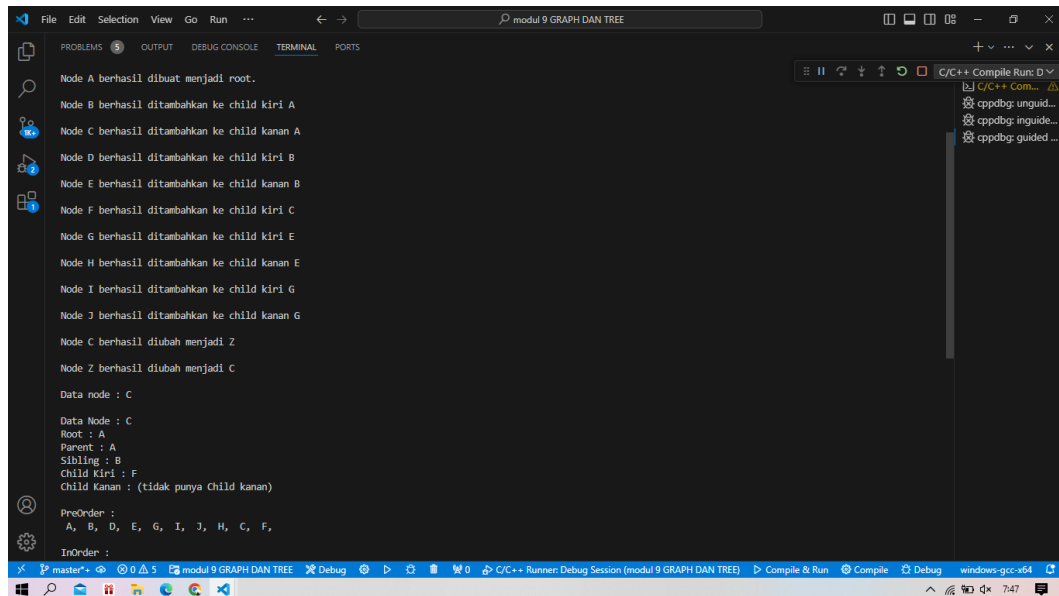
```
cout << "\nPreOrder :" << endl;

    preOrder(root);

    cout << "\n" << endl;

    characteristic();
}
```

Screenshots output:



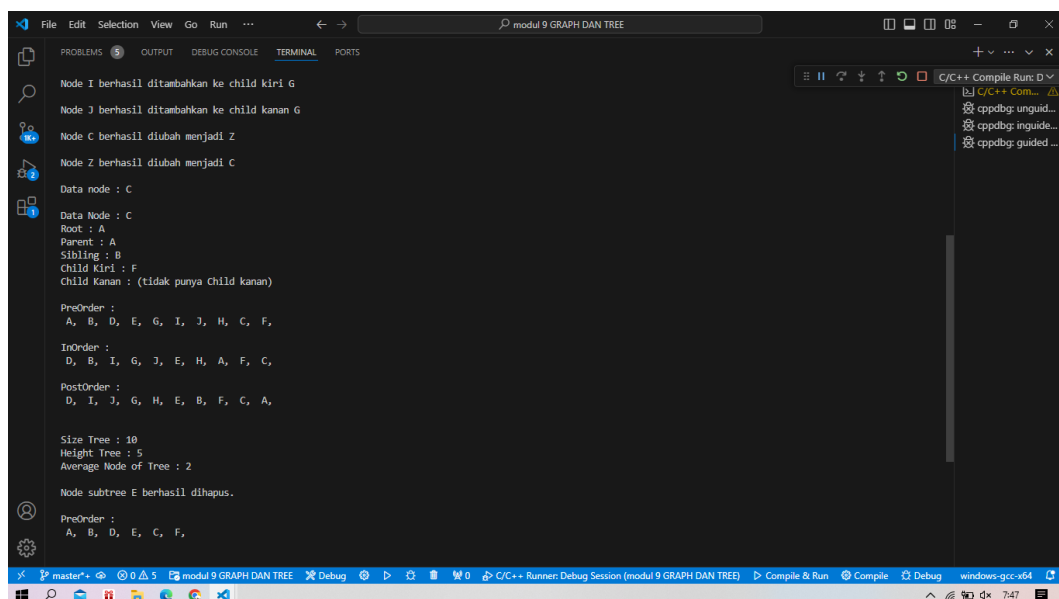
```
modul 9 GRAPH DAN TREE

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 :
```



```
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,
```

Deskripsi :

➤ Struktur Pohon:

- data: karakter untuk menyimpan data pada node.

- left, right, parent: pointer ke node anak kiri, anak kanan, dan parent masing-masing.

➤ **Variabel Global:**

- root: pointer ke node root (akar) dari pohon binary.

➤ **Fungsi-fungsi:**

- init(): menginisialisasi pohon binary dengan root bernilai NULL (kosong).
- isEmpty(): mengecek apakah pohon binary kosong.
- newPohon(char data): mengalokasikan memori untuk node baru dengan data data dan mengembalikan pointer ke node tersebut.
- buatNode(char data): membuat node baru dengan data data sebagai root pohon binary.
- insertLeft(char data, Pohon *node): menambahkan node baru sebagai anak kiri dari node node dengan data data.
- insertRight(char data, Pohon *node): menambahkan node baru sebagai anak kanan dari node node dengan data data.

- `update(char data, Pohon *node)`: memperbaharui data pada node node dengan nilai data.
- `retrieve(Pohon *node)`: menampilkan data pada node node.
- `find(Pohon *node)`: menampilkan informasi dari node node termasuk data, parent, sibling (anak lain dari parent), dan child (anak kiri dan kanan).

➤ **Penelusuran (Traversal):**

- `preOrder(Pohon *node)`: penelusuran dengan urutan root, anak kiri, anak kanan.
- `inOrder(Pohon *node)`: penelusuran dengan urutan anak kiri, root, anak kanan.
- `postOrder(Pohon *node)`: penelusuran dengan urutan anak kiri, anak kanan, root.
- `deleteTree(Pohon *node)`: menghapus seluruh node pada pohon binary secara rekursif.
- `deleteSub(Pohon *node)`: menghapus seluruh subtree (subpohon) yang berakar pada node node.
- `clear()`: menghapus seluruh node pada pohon binary.

- `size(Pohon *node)`: menghitung jumlah node pada pohon binary secara rekursif.
- `height(Pohon *node)`: menghitung tinggi dari pohon binary (jarak maksimum dari root ke node terjauh).
- `characteristic()`: menampilkan karakteristik pohon binary seperti jumlah node (`size`) dan tinggi (`height`), serta menghitung rata-rata jumlah node per level (jika tinggi pohon tidak nol).

➤ **Fungsi main():**

- Memanggil fungsi `init()` untuk menginisialisasi pohon binary.
- Membuat node root dengan data 'A'.
- Memanggil fungsi untuk menambah node-node baru ke dalam pohon binary dengan data 'B', 'C', dst.
- Memanggil fungsi `update()`, `retrieve()`, `find()` untuk memperbaharui, menampilkan, dan mencari informasi node tertentu.
- Menampilkan hasil penelusuran dengan `preOrder()`, `inOrder()`, `postOrder()`.
- Menampilkan karakteristik pohon binary dengan `characteristic()`.

- Memanggil fungsi deleteSub() untuk menghapus subtree.
- Menampilkan kembali hasil penelusuran dengan preOrder() dan karakteristik pohon binary.

c. unguided/tugas

unguided 1

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <cmath>

using namespace std;

// Deklarasi struktur untuk menyimpan informasi jarak antar kota
struct Jarak {
    string kota1, kota2;
    int jarak;
};

// Deklarasi fungsi untuk menghitung jarak Euclidean antar dua titik
int hitungJarak(int x1, int y1, int x2, int y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}
```

```
int main() {  
  
    // Meminta input jumlah kota  
    int jumlahKota;  
  
    cout << "Masukkan jumlah kota: ";  
    cin >> jumlahKota;  
  
    // Deklarasi vector untuk menyimpan informasi kota  
    vector<pair<string, pair<int, int>>> kotaList;  
  
    // Memasukkan informasi kota berdasarkan input pengguna  
    for (int i = 0; i < jumlahKota; ++i) {  
        string namaKota;  
        int x, y;  
  
        cout << "Masukkan nama kota ke-" << i + 1 << ": ";  
        cin >> namaKota;  
  
        cout << "Masukkan koordinat x kota " << namaKota << ": ";  
        cin >> x;  
  
        cout << "Masukkan koordinat y kota " << namaKota << ": ";  
        cin >> y;  
  
        kotaList.push_back({namaKota, {x, y}});  
    }  
}
```

```

// Deklarasi vector untuk menyimpan informasi jarak antar kota
vector<Jarak> jarakList;

// Memasukkan informasi jarak antar kota berdasarkan input pengguna
for (int i = 0; i < jumlahKota - 1; ++i) {
    for (int j = i + 1; j < jumlahKota; ++j) {
        string kota1 = kotaList[i].first;
        string kota2 = kotaList[j].first;

        int jarak = hitungJarak(kotaList[i].second.first, kotaList[i].second.second,
kotaList[j].second.first, kotaList[j].second.second);

        jarakList.push_back({kota1, kota2, jarak});
    }
}

// Menampilkan daftar jarak antar kota
cout << "\nDaftar Jarak Antar Kota:" << endl;
for (const Jarak& j : jarakList) {
    cout << j.kota1 << " - " << j.kota2 << ": " << j.jarak << " km" << endl;
}

// Meminta input kota asal dan kota tujuan
string kotaAsal, kotaTujuan;
cout << "\nMasukkan kota asal: ";
cin >> kotaAsal;
cout << "Masukkan kota tujuan: ";
cin >> kotaTujuan;

```

```
// Mencari jarak dari kota asal ke kota tujuan

for (const Jarak& j : jarakList) {

    if ((j.kota1 == kotaAsal && j.kota2 == kotaTujuan) || (j.kota1 == kotaTujuan &&
j.kota2 == kotaAsal)) {

        cout << "Jarak dari " << kotaAsal << " ke " << kotaTujuan << " adalah " << j.jarak
<< " km" << endl;

        break;

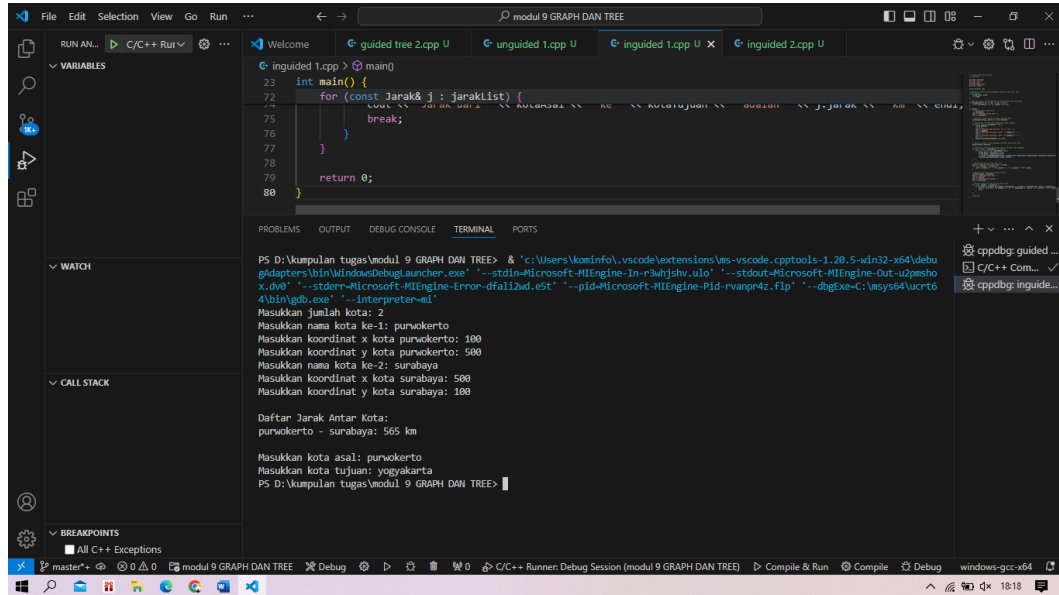
    }

}

return 0;

}
```

Screenshots output:



```
PS D:\Kumpulan tugas\modul 9 GRAPH DAN TREE> & 'c:\Users\kaminfa\.vscode\extensions\ms-vscode.cpptools-1.28.5-win32-x64\bu
g\adapters\bin\windowsdebuglauncher.exe' '--stdin=Microsoft-MIEngine-In-vsdbgshv.ulo' '--stdout=Microsoft-MIEngine-Out-udpesho
x.d66' '--stderr=Microsoft-MIEngine-Error-dfall2wd.est' '--pid=Microsoft-MIEngine-Pid-rvnp4z.flp' '--dbgExe=C:\msys64\ucrt6
4\bin\gdb.exe' '--interpreter=mi'
Masukkan jumlah kota: 2
Masukkan nama kota ke-1: purwokerto
Masukkan koordinat x kota purwokerto: 180
Masukkan koordinat y kota purwokerto: 500
Masukkan nama kota ke-2: surabaya
Masukkan koordinat x kota surabaya: 500
Masukkan koordinat y kota surabaya: 180

Daftar Jarak Antar Kota:
purwokerto - surabaya: 565 km

Masukkan kota asal: purwokerto
Masukkan kota tujuan: yogyakarta
PS D:\Kumpulan tugas\modul 9 GRAPH DAN TREE>
```

Deskripsi :

1. Meminta Input Jumlah Kota:

- Program meminta pengguna untuk memasukkan jumlah kota yang ingin dimasukkan.
- Jumlah ini disimpan dalam variabel jumlahKota.

2. Memasukkan Informasi Kota:

- Program menggunakan loop for untuk mengulangi sebanyak jumlahKota.
- Dalam setiap iterasi:

- Program meminta pengguna untuk memasukkan nama kota.
- Program meminta pengguna untuk memasukkan koordinat x kota.
- Program meminta pengguna untuk memasukkan koordinat y kota.
- Informasi kota disimpan dalam pair dan ditambahkan ke vector kotaList.

3. Memasukkan Informasi Jarak Antar Kota:

- Program menggunakan dua loop for bersarang untuk mengulangi semua kemungkinan pasangan kota.
- Dalam setiap iterasi:
 - Nama kota asal dan kota tujuan diekstrak dari kotaList.
 - Jarak Euclidean antara dua titik dihitung menggunakan fungsi hitungJarak.
 - Informasi jarak disimpan dalam struktur Jarak dan ditambahkan ke vector jarakList.

4. Menampilkan Daftar Jarak Antar Kota:

- Program mencetak header "Daftar Jarak Antar Kota:".
- Program menggunakan loop for untuk mengulangi jarakList.
- Untuk setiap jarak, program mencetak nama kota asal, nama kota tujuan, dan jaraknya dalam format "kota1 - kota2: jarak km".

5. Meminta Input Kota Asal dan Kota Tujuan:

- Program meminta pengguna untuk memasukkan nama kota asal.
- Program meminta pengguna untuk memasukkan nama kota tujuan.
- Nama kota asal dan kota tujuan disimpan dalam variabel kotaAsal dan kotaTujuan.

6. Mencari Jarak Antara Kota Asal dan Kota Tujuan:

- Program menggunakan loop for untuk mengulangi jarakList.
- Jika nama kota asal dan kota tujuan cocok dengan salah satu entri jarak, program:
 - Mencetak jarak antara kota asal dan kota tujuan dalam format "Jarak dari

kotaAsal ke kotaTujuan adalah jarak km".

- Keluar dari loop for.

7. Mengembalikan Nilai:

- Program mengembalikan nilai 0, yang menunjukkan eksekusi yang sukses.

Fungsi Penting

- `hitungJarak(x1, y1, x2, y2)`: Menghitung jarak Euclidean antara dua titik menggunakan rumus Pythagoras.

Struktur Data

- `Jarak`: Struktur untuk menyimpan informasi jarak antar kota, termasuk nama kota asal, nama kota tujuan, dan jaraknya.
- `kotaList`: Vector untuk menyimpan informasi kota, termasuk nama kota dan koordinatnya (x, y).
- `jarakList`: Vector untuk menyimpan informasi jarak antar kota, termasuk nama kota asal, nama kota tujuan, dan jaraknya.

Unguided 2

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>
using namespace std;

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

Pohon *root;

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;
    }
}

}

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);
        }
    }
}
```

```
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);  
        }  
    }  
}
```

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

```
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;  
            }  
        }  
    }  
}
```



```
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;  
    }  
}  
  
void clear() {  
    if (isEmpty())  
        cout << "\nBuat tree terlebih dahulu!!" << endl;  
    else {  
        deleteTree(root);  
        cout << "\nPohon berhasil dihapus." << endl;  
    }  
}
```

```

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);
        }
    }
}

```

```

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;
            }
        }
    }
}

```

```

}

}

}

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;
}

void displayChild(Pohon *node) {
    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
    } else {
        cout << "\nNode: " << node->data << endl;
        if (node->left)
            cout << "Child Kiri: " << node->left->data << endl;
        else
            cout << "Child Kiri: (tidak punya Child kiri)" << endl;
        if (node->right)
            cout << "Child Kanan: " << node->right->data << endl;
        else

```

```
cout << "Child Kanan: (tidak punya Child kanan)" << endl;

    }
}

void displayDescendants(Pohon *node) {
    if (!node) {
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
    } else {
        cout << "\nDescendants of Node " << node->data << ": ";
        preOrder(node);
        cout << endl;
    }
}

void menu() {
    int pilihan;
    char data;
    Pohon *node = nullptr;

    do {
```

```
cout << "\nMenu:\n";

    cout << "1. Buat Node\n";
    cout << "2. Insert Left\n";
    cout << "3. Insert Right\n";
    cout << "4. Update Node\n";
    cout << "5. Retrieve Node\n";
    cout << "6. Find Node\n";
    cout << "7. PreOrder Traversal\n";
    cout << "8. InOrder Traversal\n";
    cout << "9. PostOrder Traversal\n";
    cout << "10. Delete SubTree\n";
    cout << "11. Clear Tree\n";
    cout << "12. Display Characteristics\n";
    cout << "13. Display Child Nodes\n";
    cout << "14. Display Descendants\n";
    cout << "0. Exit\n";
    cout << "Pilih menu: ";
    cin >> pilihan;

    switch (pilihan) {
        case 1:
            cout << "Masukkan data node (karakter): ";
            cin >> data;
            buatNode(data);
            break;
        case 2:
            cout << "Masukkan data node (karakter): ";
            cin >> data;
```

```
if (root) {  
    cout << "Masukkan data parent node (karakter): ";  
    cin >> data;  
    node = insertLeft(data, root);  
} else {  
    cout << "Pohon belum dibuat!\n";  
}  
break;  
  
case 3:  
    cout << "Masukkan data node (karakter): ";  
    cin >> data;  
    if (root) {  
        cout << "Masukkan data parent node (karakter): ";  
        cin >> data;  
        node = insertRight(data, root);  
    } else {  
        cout << "Pohon belum dibuat!\n";  
    }  
    break;
```

case 4:

```
cout << "Masukkan data baru (karakter): ";  
cin >> data;  
if (root) {  
    cout << "Masukkan data node yang ingin diganti (karakter): ";  
    cin >> data;  
    update(data, root);  
} else {  
    cout << "Pohon belum dibuat!\n";  
}  
break;
```

case 5:

```
cout << "Masukkan data node (karakter) yang ingin dilihat: ";  
cin >> data;  
retrieve(root);  
break;
```

case 6:

```
cout << "Masukkan data node (karakter) yang ingin dicari: ";  
cin >> data;  
find(root);  
break;
```

case 7:

```
cout << "PreOrder Traversal: ";  
preOrder(root);  
cout << "\n";  
break;
```

case 8:

```
cout << "InOrder Traversal: ";  
inOrder(root);  
cout << "\n";  
break;
```

case 9:

```
cout << "PostOrder Traversal: ";  
postOrder(root);  
cout << "\n";  
break;
```

case 10:

```
cout << "Masukkan data node (karakter) dari subtree yang ingin dihapus: ";  
cin >> data;  
deleteSub(root);  
break;
```

case 11:

```
clear();  
break;
```

case 12:

```
characteristic();  
break;
```

case 13:

```
cout << "Masukkan data node (karakter) yang ingin dilihat child-nya: ";  
cin >> data;  
displayChild(root);  
break;
```


case 14:

```
        cout << "Masukkan data node (karakter) yang ingin dilihat descendant-nya:
";

        cin >> data;

        displayDescendants(root);

        break;

case 0:

        cout << "Keluar dari program...\n";

        break;

default:

        cout << "Pilihan tidak valid!\n";

        break;

    }

} while (pilihan != 0);

}
```

```
int main() {

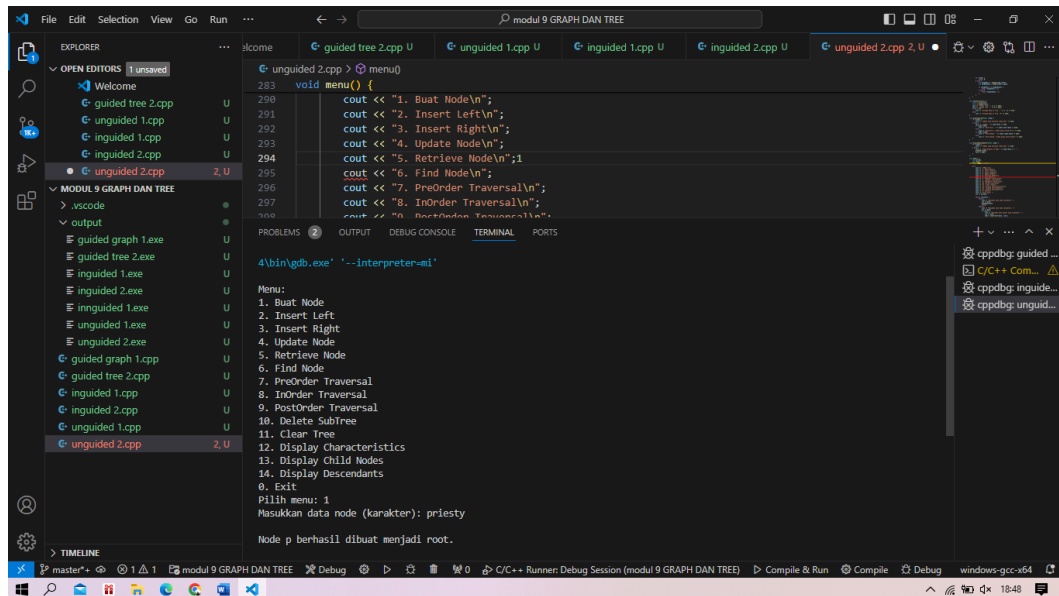
    init();

    menu();

    return 0;

}
```

Screenshots output:



Deskripsi :

➤ Struktur dan Variabel Global:

- Pohon: Struktur ini mewakili sebuah node dalam pohon biner. Ia memiliki empat anggota data:
 - data: Menyimpan data karakter dari node.
 - left: Penunjuk ke node anak kiri.
 - right: Penunjuk ke node anak kanan.
 - parent: Penunjuk ke node parent (fungsionalitas tambahan dibandingkan kode sebelumnya).
- root: Variabel penunjuk global ini menunjuk ke node root dari pohon.

➤ Fungsi:

- `init()`: Meminisiasi pohon dengan mengatur penunjuk root menjadi NULL.
- `isEmpty()`: Memeriksa apakah pohon kosong dengan memeriksa apakah root adalah NULL.
- `newPohon(char data)`: Membuat node baru dengan data yang diberikan dan menginisiasi penunjuk kiri, kanan, dan parentnya menjadi NULL.
- `buatNode(char data)`: Membuat node root baru dengan data yang diberikan jika pohon kosong. Jika tidak, ia mencetak pesan yang menunjukkan pohon sudah ada.
- `insertLeft(char data, Pohon *node)`: Memasukkan node baru sebagai anak kiri dari node tertentu. Ia memeriksa apakah anak kiri sudah ada dan menangani error.
- `insertRight(char data, Pohon *node)`: Memasukkan node baru sebagai anak kanan dari node tertentu, mirip dengan `insertLeft`.
- `update(char data, Pohon *node)`: Memperbarui data dari node tertentu. Ia memeriksa apakah node ada dan menangani error.

- `retrieve(Pohon *node)`: Mengambil data dari node tertentu. Ia memeriksa apakah node ada dan menangani error.
- `find(Pohon *node)`: Mencetak informasi tentang node tertentu, termasuk data, parent (fungsionalitas baru), sibling (jika ada), dan node anak.
- `preOrder(Pohon *node)`: Melakukan preorder traversal dari pohon dan mencetak data node.
- `inOrder(Pohon *node)`: Melakukan inorder traversal dari pohon dan mencetak data node.
- `postOrder(Pohon *node)`: Melakukan postorder traversal dari pohon dan mencetak data node.
- `deleteTree(Pohon *node)`: Menghapus seluruh pohon secara rekursif, dimulai dari node tertentu.
- `deleteSub(Pohon *node)`: Menghapus subtree yang berakar pada node tertentu.
- `clear()`: Menghapus seluruh pohon dan mengatur penunjuk root menjadi NULL.
- `size(Pohon *node)`: Menghitung ukuran (jumlah node) dari pohon secara rekursif.
- `height(Pohon *node)`: Menghitung tinggi (kedalaman) dari pohon secara rekursif.

- `characteristic()`: Mencetak ukuran, tinggi, dan rata-rata jumlah node dari pohon.

➤ **Fungsi Baru:**

- Struktur Pohon sekarang menyertakan penunjuk parent untuk melacak node parent untuk setiap node.
- Fungsi `find` menggunakan penunjuk parent untuk menampilkan informasi tentang node parent dari node yang dicari.
- Fungsi `displayChild` mengambil node sebagai input dan menampilkan node anaknya (jika ada).

➤ **Program Berbasis Menu:**

- Fungsi menu menyediakan antarmuka pengguna untuk berinteraksi dengan pohon biner.
- Pengguna dapat memilih berbagai opsi seperti membuat node, melakukan insertion, update, deletion, traversal, dan menampilkan karakteristik atau node anak.

E. Referensi

<https://etd.repository.ugm.ac.id/penelitian/detail/181993>