LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA

MODUL IX

DISUSUN OLEH :

Nama: Ibnu Rizal Mutaqim
Nim : (2311102067)

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

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

A. Dasar Teori

1. Pengertian Graph

   Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan.

   Graph secara matematis dinyatakan sebagai :

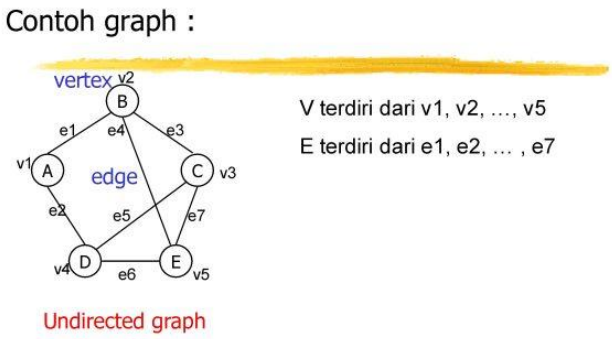
   G = (V, E)

   Dimana

   G = Graph

   V = Simpul atau Vertex, atau Node, atau Titik

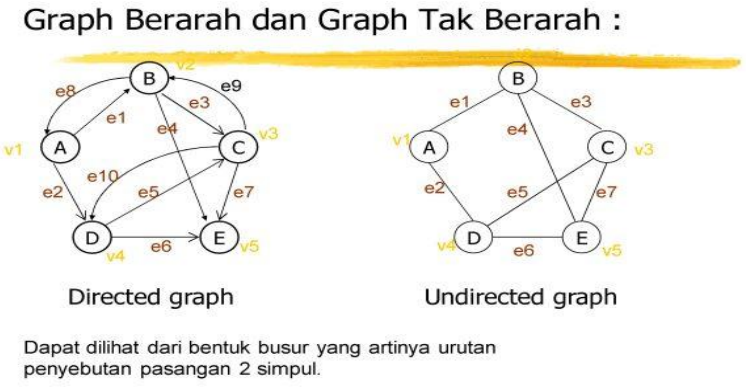
   E = Busur atau Edge, atau Arch



   Ciri-ciri graph:

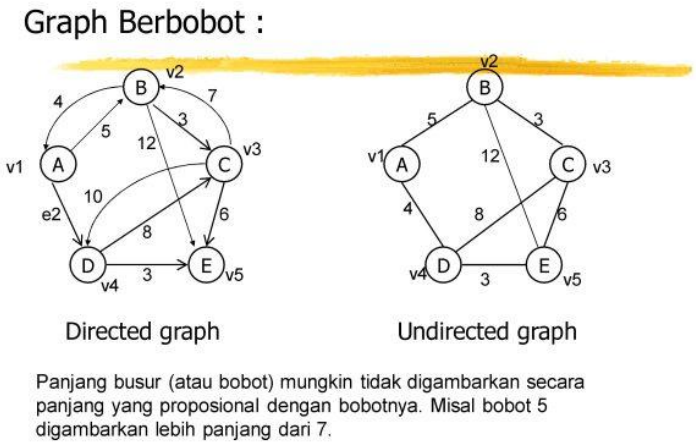
   - Sebuah graph mungkin hanya terdiri dari satu simpul
   - Sebuah graph belum tentu semua simpulnya terhubung dengan busur
   - Sebuah graph mungkin mempunyai simpul yang tak terhubung dengan simpul yang lain
   - Sebuah graph mungkin semua simpulnya saling berhubungan

Jenis-jenis Graph



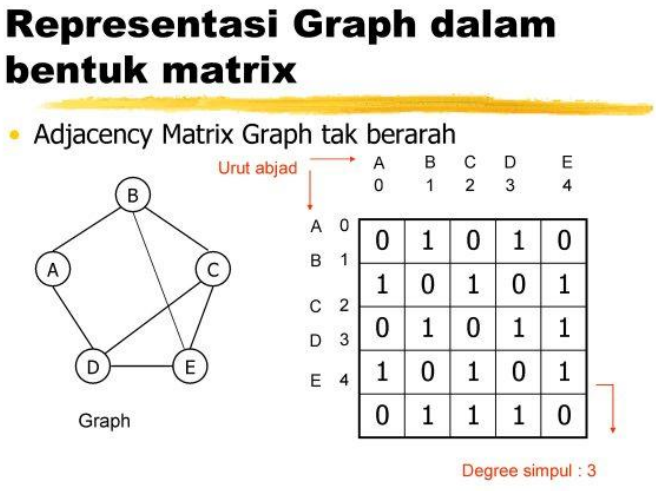
- Graph Berarah (directed graph) : Urutan simpul mempunyai arti. Mis busur AB adalah e1 sedangkan busur BA adalah e8.

- •Graph Tak Berarah (undirected graph atau non-directed graph) : Urutan simpul dalam sebuah busur tidak dipentingkan. Mis busur e1 dapat disebut busur AB atau BA.



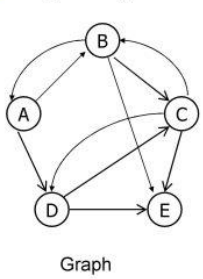
Graph Berbobot (Weighted Graph)

- Jika setiap busur mempunyai nilai yang menyatakan hubungan antara 2 buah simpul, maka busur tersebut dinyatakan memiliki bobot.
- Bobot sebuah busur dapat menyatakan panjang sebuah jalan dari 2 buah titik, jumlah rata-rata kendaraan perhari yang melalui sebuah jalan, dll.



Representasi Graph dalam bentuk matrix

- Adjacency Matrix Graph berarah



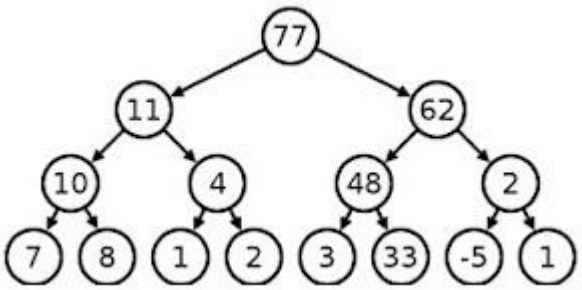
Graph

ke → / dari ↓	A 0	B 1	C 2	D 3	E 4	
A 0	0	1	0	1	0	
B 1	1	0	1	0	1	out →
C 2	0	1	0	1	1	
D 3	0	0	1	0	1	
E 4	0	0	0	0	0	
						↓ in

Representasi Graph dalam bentuk Linked List

- Adjency List graph tak berarah
- Digambarkan sebagai sebuah simpul yang memiliki 2 pointer.
- Simpul vertex : Simpul edge :

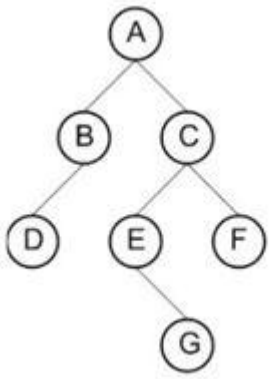
2. Tree atau Pohon
   Tree atau pohon merupakan struktur data yang tidak linear yang digunakan untuk mempresentasikan data yang bersifat hirarki antara elemen-elemennya. Definisi tree yaitu kumpulan elemen yang salah satu elemennya disebut root (akar) dan elemen yang lain disebut simpul (node) yang terpecah menjadi sejumlah kumpulan yang tidak saling berhubungan satu sama lain yang disebut sub-tree atau cabang.



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

Binary Tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut, maka tiap node dalam binary tree hanya boleh memiliki paling banyak dua child. contoh implementasi binary tree.



Operasi pada tree

a) Create
   Create digunakan untuk membentuk binary tree baru yang masih kosong.
b) Clear
   Clear digunakan untuk mengosongkan binary tree yang sudah ada.
c) Empty
   Empty digunakan untuk memeriksa apakah binary tree masih kosong.
d) Insert
   Insert digunakan untuk memasukkan sebuah node ke dalam tree.
e) Find
   Find digunakan untuk mencari root, parent, left child , atau right child dari suatu node dengan syarat tree tidak boleh kosong.
f) Update
   Update digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer

current dengan syarat tree tidak boleh kosong.

g) Retrieve
   Retrieve digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
h) Delete Sub
   DeleteSub digunakan untuk menghapus sebuah sub-tree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
i) Characteristic
   Characteristic digunakan untuk mengetahui karakteristik dari suatu tree, yakni size, height, serta average length-nya.
j) Traverse
   Traverse digunakan untuk mengunjungi seluruh node-node pada tree, masing-masing sekali.

Tree Traversal

Tree traversal merupakan sebuah kunjungan yang berawal dari root, mengunjungi setiap node dalam tree masing-masing sekali. Kunjungan atau traversing dapat dilakukan dengan 3 cara yaitupre order, in order dan post order. [10]

a. Preorder

Algoritma preorder :

- Mencetak info pada node yang dikunjungi.
- Mengunjungi cabang kiri.
- Mengunjungi cabang kanan.

b. Inorder

Algoritma inorder :

- Mengunjungi cabang kiri.
- Mencetak info pada node yang dikunjungi.
- Mengunjungi cabang kanan.

c. Postorder

Algoritma postorder :

- Mengunjungi cabang kiri.
- Mengunjungi cabang kanan.
- Mencetak info pada node yang dikunjungi.
- Breadth-first search (BFS) dan Depth-First-Search (DFS)

B. Guided

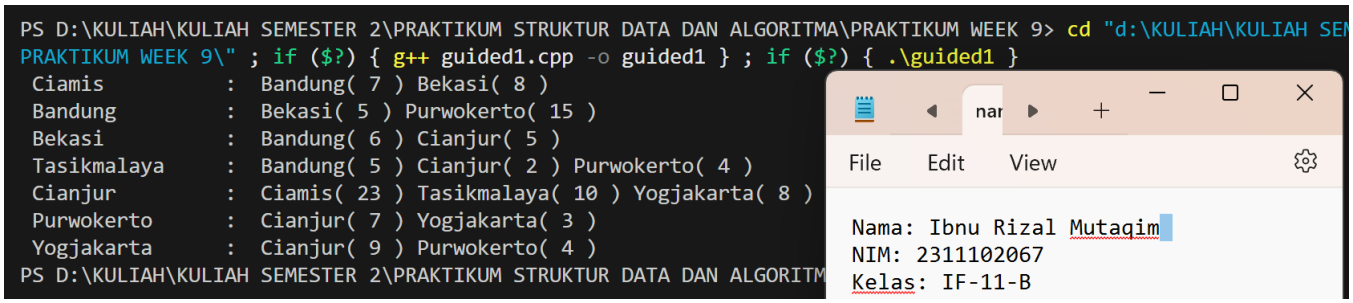
Guided 1

Sourcode :

```
#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;
}
```

Output :



Deskripsi :

Kode tersebut adalah implementasi sederhana dari graf , yang menggambarkan hubungan antara kota-kota di Indonesia dan jarak antara kota-kota tersebut. Variabel `simpul` menyimpan nama-nama kota, sedangkan matriks `busur` menyimpan bobot jarak antara kota-kota. Fungsi `tampilgraph()` digunakan untuk menampilkan graf dalam bentuk matriks, menunjukkan kota-kota yang terhubung dan jaraknya. Dalam `main()`, graf ditampilkan menggunakan fungsi `tampilgraph()`. Dengan kode ini, kita dapat dengan mudah melihat dan menganalisis jarak antara kota-kota dalam bentuk graf.

Guided 2

Source Code:

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
```

```
void init()
{
   root = NULL;
}
// Cek Node
int isEmpty()
{
   if (root == NULL)
      return 1; // true
   else
      return 0; // false
}
// Buat Node Baru
void buatNode(char data)
{
   if (isEmpty() == 1)
   {
      root = new Pohon();
      root->data = data;
      root->left = NULL;
      root->right = NULL;
      root->parent = NULL;
      cout << "\n Node " << data << " berhasil dibuat menjadi root." << endl;
   }
   else
   {
      cout << "\n Pohon sudah dibuat" << endl;
   }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
   if (isEmpty() == 1)
   {
```

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

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

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

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

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

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

```
        }
    }
}
// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil dihapus." << endl;
    }
}
// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
```

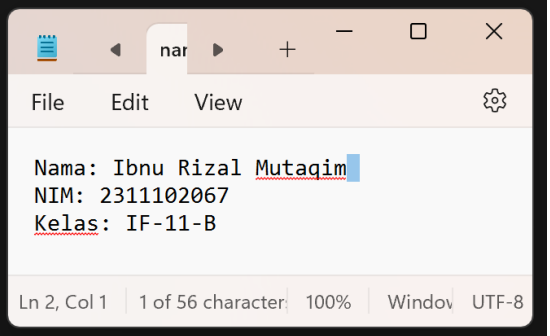
```
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}
// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!"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

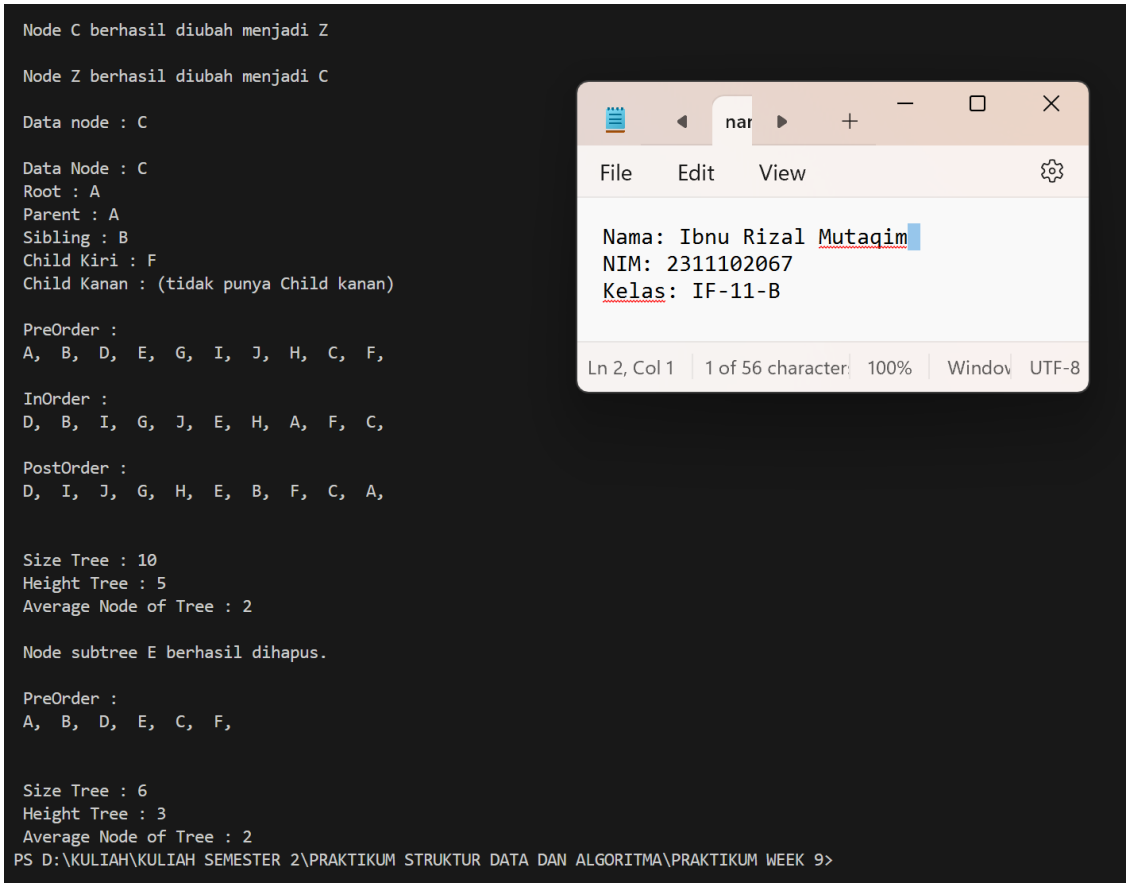
```
        else
        {
            return heightKanan + 1;
        }
      }
   }
}
// Karakteristik Tree
void charateristic()
{
   cout << "\n Size Tree : " << size() << endl;
   cout << " Height Tree : " << height() << endl;
   cout << " Average Node of Tree : " << size() / height() << endl;
}
int main()
{
   buatNode('A');
   Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
      *nodeI, *nodeJ;
   nodeB = insertLeft('B', root);
   nodeC = insertRight('C', root);
   nodeD = insertLeft('D', nodeB);
   nodeE = insertRight('E', nodeB);
   nodeF = insertLeft('F', nodeC);
   nodeG = insertLeft('G', nodeE);
   nodeH = insertRight('H', nodeE);
   nodeI = insertLeft('I', nodeG);
   nodeJ = insertRight('J', nodeG);
   update('Z', nodeC);
   update('C', nodeC);
   retrieve(nodeC);
   find(nodeC);
   cout << "\n PreOrder :" << endl;
   preOrder(root);
```

```
    cout << "\n" << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n" << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n" << endl;
    charateristic();
}
```

Output:

```
PS D:\KULIAH\KULIAH SEMESTER 2\PRAKTIKUM STRUKTUR DATA DAN ALGORITMA\PRAKTIKUM WEEK 9> cd "d:\KULIAH\KULIAH SEMEST
 -o guided_2 } ; if ($?) { .\guided_2 }

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
```





Deskripsi:

Kode tersebut adalah implementasi pohon biner. Ini memungkinkan pembuatan, penambahan, pembaruan, penelusuran, dan penghapusan node dalam pohon. Fungsi-fungsi ini mencakup inisialisasi, pengecekan kosong, pembuatan node baru, penambahan node anak, serta penelusuran pre-order, in-order, dan post-order. Selain itu, ada fungsi untuk menghapus pohon atau subpohon, menghitung ukuran dan tinggi pohon, serta menampilkan karakteristiknya. Ini memberikan alat yang lengkap untuk manipulasi dan analisis struktur data pohon biner dalam program.

C. Unguided

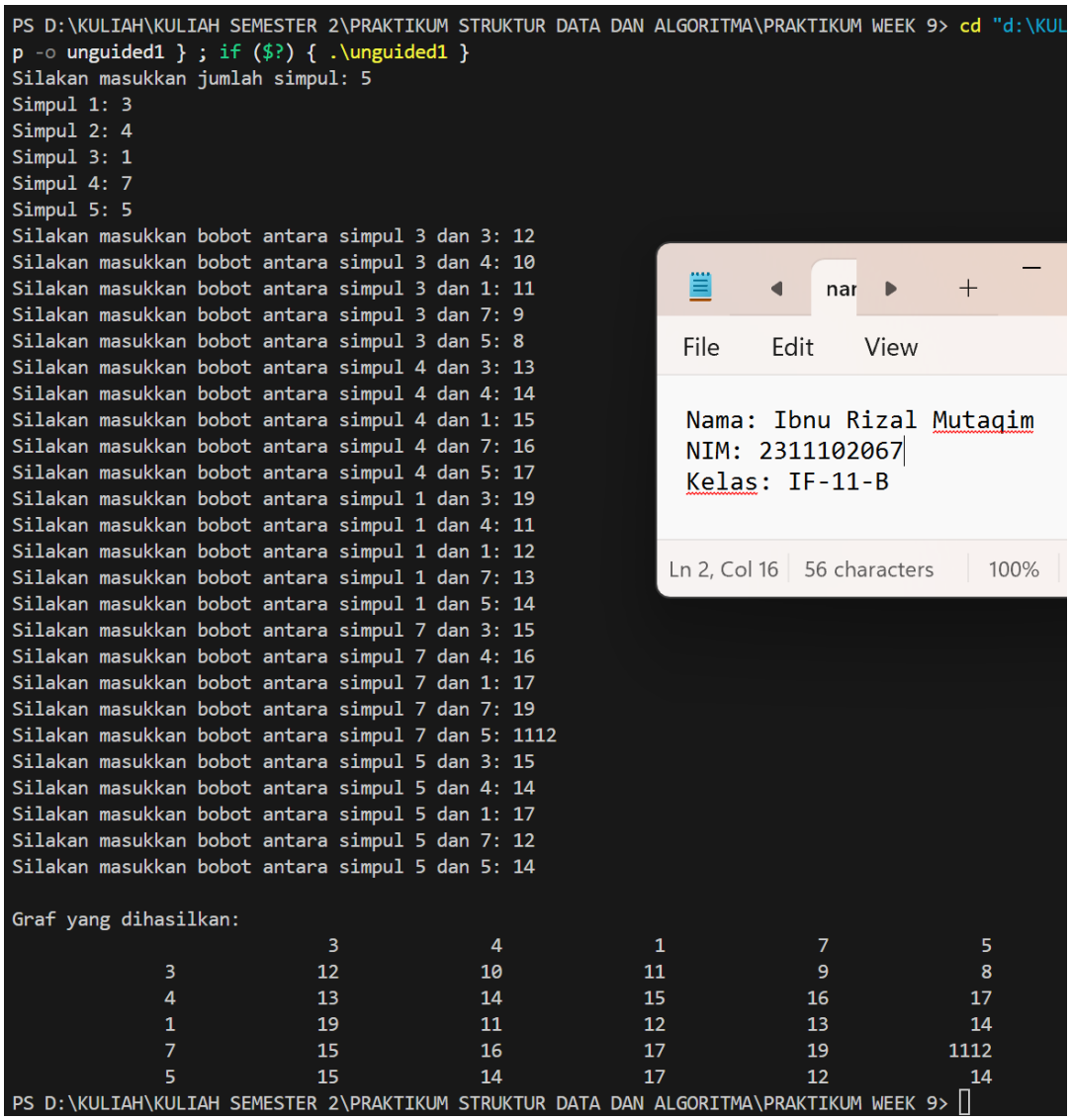
Unguided 1

Sourcode

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

int main() {
    int jumlahsimpul1_2311102067;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahsimpul1_2311102067;

    string simpul[jumlahsimpul1_2311102067];
    int busur[jumlahsimpul1_2311102067][jumlahsimpul1_2311102067];
```

```
    for (int i = 0; i < jumlahsimpul1_2311102067; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jumlahsimpul1_2311102067; i++) {
        for (int j = 0; j < jumlahsimpul1_2311102067; 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 < jumlahsimpul1_2311102067; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < jumlahsimpul1_2311102067; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jumlahsimpul1_2311102067; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Output:



Deskripsi :

Program ini adalah program yang memungkinkan pengguna memasukkan informasi tentang sebuah graf. Pertama, pengguna diminta untuk memasukkan jumlah simpul dalam graf. Program kemudian membuat array untuk menyimpan nama setiap simpul dan array dua dimensi untuk menyimpan bobot antara setiap pasangan simpul. Setelah itu, pengguna memasukkan nama untuk setiap simpul dan bobot untuk setiap pasangan simpul. Setelah semua masukan diterima, program akan menampilkan graf yang dihasilkan dengan judul simpul di baris pertama dan bobot antara simpul-simpul di baris berikutnya. Dengan demikian, program ini memudahkan pengguna untuk memasukkan informasi tentang sebuah graf dan menampilkannya secara visual untuk analisis lebih lanjut.

Unguided 2

Sourcode

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

// Deklarasi Pohon
```

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

// Pointer global
Pohon *root_2311102067;

// Inisialisasi
void init() {
    root_2311102067 = NULL;
}

bool isEmpty() {
    return root_2311102067 == 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_2311102067 = 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 memiliki child kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child kiri dari " << 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 memiliki child kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke child kanan dari " << 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_2311102067->data << endl;

            if (!node->parent)
                cout << "Parent : (tidak memiliki 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 memiliki sibling)" << endl;

            if (!node->left)
                cout << "Child Kiri : (tidak memiliki child kiri)" << endl;
            else
                cout << "Child Kiri : " << node->left->data << endl;

            if (!node->right)
                cout << "Child Kanan : (tidak memiliki 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_2311102067) {
                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_2311102067) {
                delete root_2311102067;
                root_2311102067 = 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_2311102067);
    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) {
```

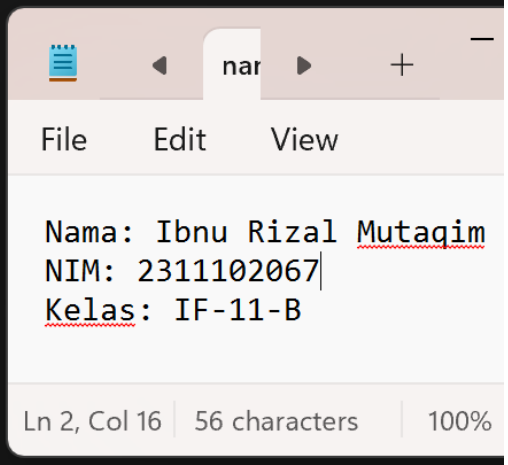
```
            return 0;
        } else {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// 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_2311102067);
    int h = height(root_2311102067);
    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;
}

// Menampilkan Child dari Sebuah Node
void displayChild(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "\nChild dari node " << node->data << " adalah:";
            if (node->left) {
                cout << " " << node->left->data;
            }
            if (node->right) {
                cout << " " << node->right->data;
            }
            cout << endl;
```

```
        }
    }
}

// Menampilkan Descendant dari Sebuah Node
void displayDescendant(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        } else {
            cout << "\nDescendant dari node " << node->data << " adalah:";
            // Gunakan rekursi untuk mencetak descendant
            if (node->left) {
                cout << " " << node->left->data;
                displayDescendant(node->left);
            }
            if (node->right) {
                cout << " " << node->right->data;
                displayDescendant(node->right);
            }
            cout << endl;
        }
    }
}

int main() {
    init();
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI, *nodeJ;

    nodeB = insertLeft('B', root_2311102067);
    nodeC = insertRight('C', root_2311102067);
    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_2311102067);
    cout << "\n" << endl;
    cout << "InOrder :" << endl;
    inOrder(root_2311102067);
    cout << "\n" << endl;
    cout << "PostOrder :" << endl;
    postOrder(root_2311102067);
    cout << "\n" << endl;
```

```
    characteristic();
    displayChild(nodeE);
    displayDescendant(nodeB);
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root_2311102067);
    cout << "\n" << endl;
    characteristic();
}
```

Output:

```
PS D:\KULIAH\KULIAH SEMESTER 2\PRAKTIKUM STRUKTUR DATA DAN ALGORITMA\PRAKTIKUM WEEK 9> cd "d:\KULIAH\KULIAH SEMEST

Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri dari A

Node C berhasil ditambahkan ke child kanan dari A

Node D berhasil ditambahkan ke child kiri dari B

Node E berhasil ditambahkan ke child kanan dari B

Node F berhasil ditambahkan ke child kiri dari C

Node G berhasil ditambahkan ke child kiri dari E

Node H berhasil ditambahkan ke child kanan dari E

Node I berhasil ditambahkan ke child kiri dari G

Node J berhasil ditambahkan ke child kanan dari 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 memiliki 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

Child dari node E adalah: G H

Descendant dari node B adalah: D
Descendant dari node D adalah:
 E
Descendant dari node E adalah: G
Descendant dari node G adalah: I
Descendant dari node I adalah:
 J
Descendant dari node J adalah:

 H
Descendant dari node H adalah:



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:\KULIAH\KULIAH SEMESTER 2\PRAKTIKUM STRUKTUR DATA DAN ALGORITMA\PRAKTIKUM WEEK 9>
```



Deskripsi :

Program ini adalah implementasi pohon biner yang mendefinisikan struktur `Pohon` dengan tiga pointer (`left`, `right`, `parent`) dan satu variabel data (`data`). Pointer global `root_2311102067` menunjuk ke akar pohon. Fungsi `init()` menginisialisasi pohon, sementara `isEmpty()` memeriksa apakah pohon kosong. Fungsi `newPohon(char data)` membuat node baru, dan `buatNode(char data)` membuat node sebagai akar jika pohon kosong. Node baru dapat ditambahkan sebagai anak kiri atau kanan menggunakan fungsi `insertLeft` dan `insertRight`. Fungsi `update` memperbarui data node, sedangkan `retrieve` dan `find` menampilkan informasi node. Program menyediakan fungsi traversal: `preOrder`, `inOrder`, dan `postOrder` untuk penelusuran pohon. Untuk penghapusan, fungsi `deleteTree` menghapus seluruh pohon, `deleteSub` menghapus subtree, dan `clear()` menghapus pohon secara keseluruhan. Fungsi `size` menghitung jumlah node, `height` menghitung tinggi pohon, dan `characteristic` menampilkan ukuran, tinggi, dan rata-rata node pohon. Fungsi `displayChild` dan `displayDescendant` menampilkan anak dan keturunan node tertentu. Dalam fungsi utama (`main`), program menginisialisasi pohon, menambahkan node, melakukan operasi update, retrieve, find, traversal, menampilkan karakteristik pohon, menghapus subtree, dan menampilkan pohon setelah penghapusan.

Kesimpulan

Graph dan tree adalah dua struktur data penting. Graf terdiri dari simpul dan sisi, digunakan untuk merepresentasikan jaringan seperti jalur transportasi atau hubungan antar individu. Implementasinya dapat dilakukan dengan matriks ketetanggaan atau daftar ketetanggaan. Pohon adalah jenis graf tanpa siklus dengan satu simpul akar, di mana setiap simpul kecuali akar memiliki satu induk dan dapat memiliki anak. Implementasi pohon melibatkan struktur node dengan data dan pointer ke anak kiri, anak kanan, serta induk, serta operasi seperti penambahan node, penelusuran (pre-order, in-order, post-order), dan penghapusan node atau subtree. Memahami struktur dan operasi dasar ini memungkinkan penggunaan graf dan pohon untuk berbagai aplikasi praktis dan teoretis.

Referensi

Asisten Pratikum “Graph dan Pohon”, Learning Management System, 2024.

Hadari Blog,” Graf (Graph) dan Pohon (Tree) - Algoritma Pemrograman 2” 2019.

Graf (Graph) dan Pohon (Tree) - Algoritma Pemrograman 2 (ahmadhadari77.blogspot.com)

Ahmad Zipur,” Struktur Data : Graph – Pengertian, Jenis, Dan Algoritma”2023.

#8 Struktur Data : Graph – Pengertian, Jenis, dan Algoritma – Ahmad Zipur