

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 9
“GRAPH DAN TREE”**



**DISUSUN OLEH :
RIZKY PERLINTA SEMBIRNG
2311102061**

**DOSEN
WAHYU ANDI SAPUTRA, S.PD., M.PD.**

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

A. DASAR TEORI

1. PENGERTIAN GRAPH

Graf adalah kumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). Graph dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graph adalah dengan menyatakan objek sebagai noktah, bulatan atau titik (Vertex), sedangkan hubungan antara objek dinyatakan dengan garis (Edge).

$$G = (V, E)$$

Dimana :

G = Graph

V = Simpul atau Vertex, atau Node, atau Titik

E = Busur atau Edge, atau arc.

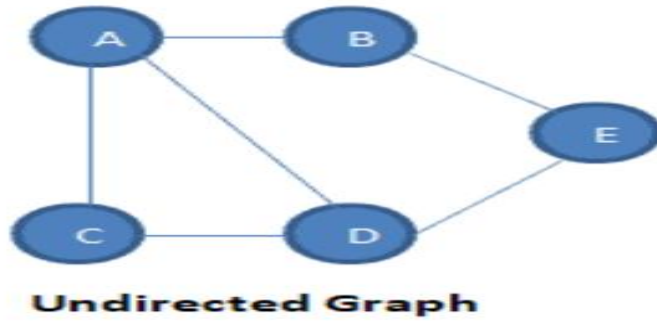
Graf merupakan suatu cabang ilmu yang memiliki banyak terapan. Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Seringkali graf digunakan untuk merepresentasikan suatu jaringan. Misalkan jaringan jalan raya dimodelkan graf dengan kota sebagai simpul (vertex/node) dan jalan yang menghubungkan setiap kotanya sebagai sisi (edge) yang bobotnya (weight) adalah panjang dari jalan tersebut.

Ada beberapa cara untuk menyimpan graph di dalam sistem komputer. Struktur data bergantung pada struktur graph dan algoritma yang digunakan untuk memanipulasi graph. Secara teori salah satu dari keduanya dapat dibedakan antara struktur list dan matriks, tetapi dalam penggunaannya struktur terbaik yang sering digunakan adalah kombinasi keduanya.

- JENIS-JENIS GRAPH DAN CONTOHNYA

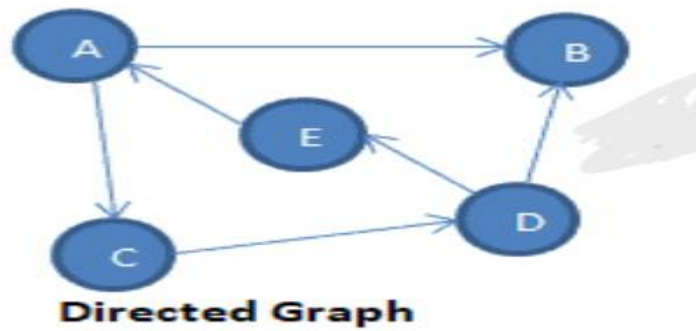
1. Graph tak berarah (undirected graph atau non-directed graph).

Contoh graph tak berarah



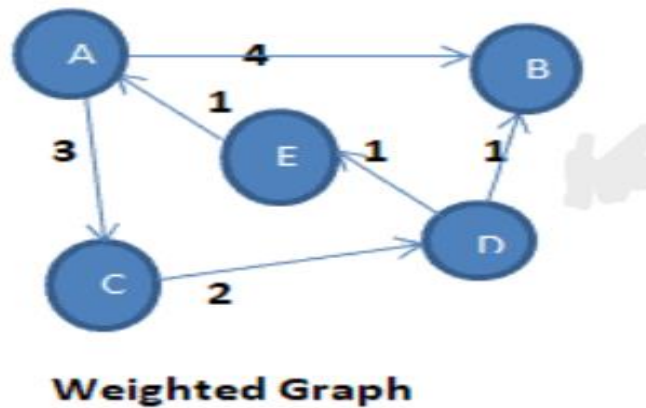
GAMBAR 1. CONTOH GAMBAR DARI UNDIRECTED GRAPH

2. Graph berarah (directed graph).



GAMBAR 2. CONTOH GAMBAR DARI DIRECTED GRAPH

3. Graph Berbobot (Weighted Graph)



GAMBAR 3. CONTOH GAMBAR DARI WEIHTED GRAPH

- **ISTIAH-ISTILAH DALAM GRAPH**

Terdapat istilah – istilah yang berkaitan dengan graph, yaitu :

a. Vertex

Vertex adalah himpunan node/titik pada sebuah graph.

b. Edge

Edge adalah garis yang menghubungkan tiap node/vertex.

c. Adjacent

Adjacent adalah dua buah titik dikatakan berdekatan jika dua buah titik tersebut terhubung dengan sebuah sisi.

d. Weight

Sebuah graph dikatakan berbobot apabila terdapat sebuah fungsi bobot bernilai real pada himpunan Edge.

e. Path

Path adalah walk dengan setiap vertex berbeda. Walk didefinisikan sebagai urutan vertex dan edge. Diawali dengan origin vertex dan diakhiri dengan terminus vertex.

f. Cycle

Cycle adalah siklus atau sirkuit yang berawal dan berakhir pada simpul yang Sama.

- **REPRESENTASI GRAPH**

Dalam pemrograman, agar data yang ada dalam graph dapat diolah, maka graph harus dinyatakan dalam suatu struktur data yang dapat mewakili graph tersebut. Dalam hal ini graph perlu direpresentasikan kedalam bentuk array dan dimensi yang sering disebut matrix atau direpresentasikan dalam bentuk linked list. Bentuk mana yang dipilih biasanya tergantung kepada efisiensi dan kemudahan dalam membuat program.

2. PENGERTIAN TREE (POHON)

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya. Tree juga adalah suatu graph yang acyclic, simple, connected yang tidak mengandung loop. Sebuah binary search tree (bst) adalah sebuah pohon biner yang boleh kosong, dan setiap nodenya harus memiliki identifier/value. Value pada semua node subpohon sebelah kiiri adalah selalu lebih kecil dari value dari root, sedangkan value subpohon di sebelah kanan adalah sama atau lebih besar dari value pada root, masing-masing subpohon tersebut (kiri dan kanan) itu sendiri adalah juga binary search tree. Struktur data bst sangat penting dalam struktur pencarian, misalkan dalam kasus pencarian dalam sebuah list, jika list sudah dalam keadaan terurut maka proses pencarian akan semakin cepat, jika kita menggunakan list contigue dan melakukan pencarian biner, akan tetapi jika kita ingin melakukan perubahan isi list (insert atau delete), menggunakan list contigue akan sangat lambat, karena prose insert dan delete dalam list contigue butuh memindahkan linked-list, yang untuk operasi insert atau delete tinggal mengatur- atur pointer, akan tetapi pada n-linked list, kita tidak bisa melakukan pointer sembarangan setiap saat, kecuali hanya satu kali dengan kata lain hanya secara sequential.

- ISTILAH-ISTILAH DALAM TREE

Berikut ini merupakan istilah yang berhubungan dengan tree.

a. Predesesor

Node yang berada diatas node tertentu. (contoh : B predesesor dari E dan F)

b. Succesor

Node yang berada dibawah node tertentu. (contoh : E dan F merupakan succesor dari B)

c. Ancestor

Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama. (contoh : A dan B merupakan ancestor dari F)

d. Descendant

Seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama. (contoh : F dan B merupakan ancestor dari A)

e. Parent

Predecessor satu level diatas satu node. (contoh : B merupakan parent dari F)

f. Child

Successor satu level dibawah satu node. (contoh : F merupakan child dari B)

g. Sibling

Node yang memiliki parent yang sama dengan satu node (contoh : E dan F adalah sibling)

h. Subtree

Bagian dari tree yang berupa suatu node beserta descendant-nya (contoh : Subtree B, E, F dan Subtree D, G, H)

i. Size

Banyaknya node dalam suatu tree (contoh : gambar tree diatas memiliki size = 8)

j. Height

Banyaknya tingkat/level dalam suatu tree (contoh : gambar tree diatas memiliki height = 3)

k. Root (Akar)

Node khusus dalam tree yang tidak memiliki predecessor (Contoh : A)

l. Leaf (Daun)

Node-node dalam tree yang tidak memiliki daun (contoh : Node E,F,C,G,H)

m. Degree (Derajat)

Banyaknya child yang dimiliki oleh suatu node (contoh : Node A memiliki derajat 3, node B memiliki derajat 2)

- **BINARY TREE**

Binary Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Dalam tree terdapat jenis-jenis tree yang memiliki sifat khusus, diantaranya adalah binary tree.

Binary tree adalah suatu tree dengan syarat bahwa tiap node (simpul) hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Tiap node dalam binary tree boleh memiliki paling banyak dua child (anak simpul), secara khusus anaknya dinamakan kiri dan kanan.

Binary Tree merupakan himpunan vertex-vertex yang terdiri dari 2 subtree (dengan disjoint) yaitu subtree kiri dan subtree kanan. Setiap vertex dalam binary tree mempunyai derajat keluar $\max = 2$.

Sebuah pohon biner adalah grafik asiklis yang terhubung dimana setiap tingkatan dari susut tidak lebih dari 3. Ini dapat ditunjukkan bahwa dalam pohon biner manapun, terdapat persis dua atau lebih simpul dengan tingkat satu daripada yang terdapat dengan tingkat tiga, tetapi bisa terdapat angka apa saja dari simpul dengan tingkat dua. Sebuah pohon biner berakar merupakan sebuah grafik yang mempunyai satu dari sudutnya dengan tingkat tidak lebih dari dua sebagai akar.

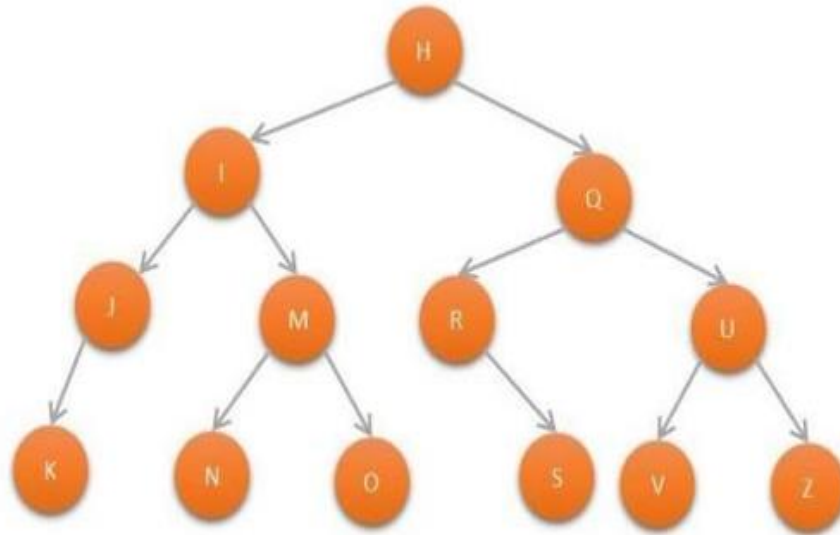
- **ISTILAH-ISTILAH DALAM BINARY TREE**

Berikut ini merupakan istilah – istilah yang berhubungan dengan binary tree:

- a. Full Binary Tree, semua simpul (kecuali daun) memiliki dua anak dan tiap cabang memiliki panjang ruas yang sama.
- b. Complete Binary Tree, Hampir sama dengan Full Binary Tree, semua simpul (kecuali daun) memiliki dua anak tetapi memiliki panjang ruas berbeda.
- c. Similar Binary Tree, dua pohon yang memiliki struktur yang sama tetapi informasinya berbeda.
- d. Equivalent binary tree, dua pohon yang memiliki struktur informasi yang sama.
- e. Kewed tree, dua pohon yang semua simpulnya mempunyai satu anak atau

turunan kecuali daun.

- CONTOH DARI BINARY TREE



Gambar 1 Struktur Data Binary Tree

GAMBAR 4. CONTOH GAMBAR DARI BINARY TREE

B. Guided

1. Program Graph

Source Code:

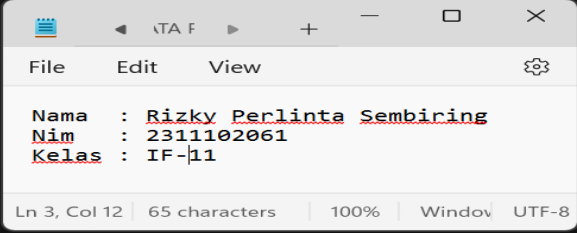
```
#include <iostream>
#include <iomanip>
using namespace std;
string
simpul[7]={"Ciamis","Bandung","Bekasi","Tasikmalaya","Cianjur","Purwokerto","Yogyakarta"};
int busur[7][7]={
    {0,7,8,0,0,0,0},
    {0,0,5,0,0,15,0},
    {0,6,0,0,5,0,0},
    {0,5,0,0,2,4,0},
    {23,0,0,10,0,0,8},
    {0,0,0,0,7,0,3},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

        }
        cout<<endl;
    }
}

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

Screenshot Output:

```
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 E:\Semester 2\Pratikum struktur data\guided>
```



The screenshot shows a Notepad window with the following text:

```
Nama : Rizky Perlinta Sembiring
Nim  : 2311102061
Kelas : IF-11
```

The status bar at the bottom of the Notepad window indicates: Ln 3, Col 12 | 65 characters | 100% | Window | UTF-8.

Deskripsi Program: Program diatas adalah program graph untuk menentukan jarak antar kota. Dari output diatas atas, kita dapat melihat bahwa setiap baris menunjukkan sebuah simpul (kota) dan busur-busur yang keluar dari simpul tersebut beserta bobotnya. Contohnya, baris pertama menunjukkan bahwa dari "Ciamis" terdapat busur menuju "Bandung" dengan bobot 7 dan menuju "Bekasi" dengan bobot 8.

2. Program Tree

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;
            }
        }
    }
}

```

```

        {
            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 characteristic()
{
    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();
}

```

Source Code:

```

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

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,

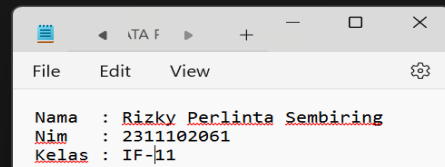
InOrder :
D, B, I, G, J, E, H, A, F, C,

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.

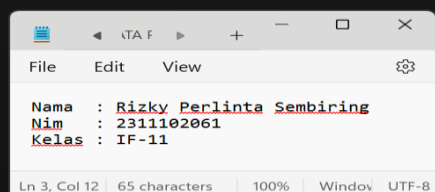
```



```

File Edit View
Nama : Rizky Perlinta Sembiring
Nim : 2311102061
Kelas : IF-11

```



```

File Edit View
Nama : Rizky Perlinta Sembiring
Nim : 2311102061
Kelas : IF-11
Ln 3, Col 12 | 65 characters | 100% | Window | UTF-8

```

```
PreOrder :  
A, B, D, E, C, F,  
  
Size Tree : 6  
Height Tree : 3  
Average Node of Tree : 2  
PS E:\Semester 2\Pratikum struk
```

File Edit View

Nama : Rizky Perlenta Sembiring
Nim : 2311102061
Kelas : IF-11

Ln 3, Col 12 | 65 characters | 100% | Window | UTF-8

Deskripsi Program: Program di atas adalah implementasi tree yang berjenis binary tree. Program ini mencakup berbagai fungsi untuk membuat, memanipulasi, dan mengelola binary tree. Fungsi-fungsi tersebut meliputi inisialisasi pohon, pengecekan apakah pohon kosong, pembuatan node baru, penambahan node sebagai anak kiri atau kanan, pengubahan data node, pengambilan data node, pencarian data node beserta informasinya, penelusuran pohon (pre-order, in-order, dan post-order), penghapusan node dan subtree, serta pengecekan ukuran dan tinggi pohon. Selain itu, program ini juga menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node per level. Fungsi Utama pada program tersebut adalah untuk mengilustrasikan penggunaan berbagai fungsi ini dengan membuat pohon biner, menambahkan node, mengubah data, menampilkan data dan informasi node, serta menampilkan hasil penelusuran dan karakteristik pohon sebelum dan sesudah penghapusan subtree.

C. UNGUIDED

1. Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

Output Program

```
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukkan bobot antar simpul
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

      BALI      PALU
BALI    0        3
PALU    4        0

Process returned 0 (0x0)   execution time : 11.763 s
Press any key to continue.
```

Source Code:

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

int main() {
    int jmlhsmp1_2311102061;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jmlhsmp1_2311102061;

    string simpul[jmlhsmp1_2311102061];
    int busur[jmlhsmp1_2311102061][jmlhsmp1_2311102061];

    for (int i = 0; i < jmlhsmp1_2311102061; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    for (int i = 0; i < jmlhsmp1_2311102061; i++) {
        for (int j = 0; j < jmlhsmp1_2311102061; 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 < jmlhsmpl_2311102061; i++) {
        cout << setw(15) << simpul[i];
    }
    cout << endl;

    for (int i = 0; i < jmlhsmpl_2311102061; i++) {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jmlhsmpl_2311102061; j++) {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Screenshot Output:

The screenshot shows the execution of a C++ program. The user is prompted to enter the number of nodes (2), then the names of the nodes (Medan and Jakarta), and finally the weights of the edges between them. The output displays the resulting graph structure as an adjacency matrix.

```

Silakan masukkan jumlah simpul: 2
Simpul 1: Medan
Simpul 2: Jakarta
Silakan masukkan bobot antara simpul Medan dan Medan: 0
Silakan masukkan bobot antara simpul Medan dan Jakarta: 100
Silakan masukkan bobot antara simpul Jakarta dan Medan: 100
Silakan masukkan bobot antara simpul Jakarta dan Jakarta: 0

Graf yang dihasilkan:

```

	Medan	Jakarta
Medan	0	100
Jakarta	100	0

Below the output, a Windows File Explorer window is open, showing the file path and details of a file named "Rizky Perlinta Sembiring".

Deskripsi Program: Program diatas adalah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya. Di program tersebut user disuruh untuk menginputkan jumlah simpul.setelah itu membuat nama simpul dan bobot simpul. Setelah selesai graph akan terbentuk.

2. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinput kan!

Source Code:

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

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

// Pointer global
Pohon *root_2311102061;

// Inisialisasi
void init() {
    root_2311102061 = NULL;
}

bool isEmpty() {
    return root_2311102061 == 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_2311102061 = 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_2311102061->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_2311102061) {
                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_2311102061) {
                delete root_2311102061;
                root_2311102061 = 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_2311102061);
        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_2311102061);
    int h = height(root_2311102061);
    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_2311102061);
    nodeC = insertRight('C', root_2311102061);
    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_2311102061);
    cout << "\n" << endl;
    cout << "InOrder :" << endl;
    inOrder(root_2311102061);
    cout << "\n" << endl;
    cout << "PostOrder :" << endl;
    postOrder(root_2311102061);
    cout << "\n" << endl;
    characteristic();
    displayChild(nodeE);
    displayDescendant(nodeB);
    deleteSub(nodeE);
    cout << "\nPreOrder :" << endl;
    preOrder(root_2311102061);
    cout << "\n" << endl;
    characteristic();
}

```

Screenshot Output:

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

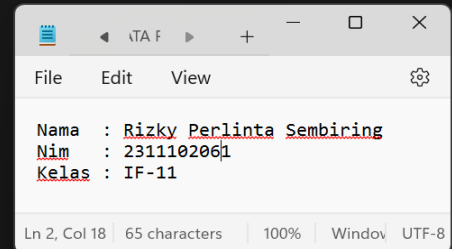
Node H berhasil ditambahkan ke child kanan dari E

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)

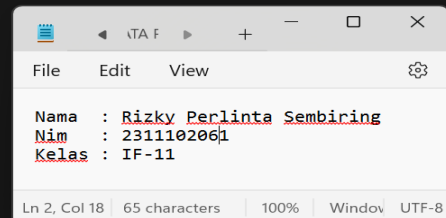
PreOrder :
 A, B, D, E, G, I, J, H, C, F,

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

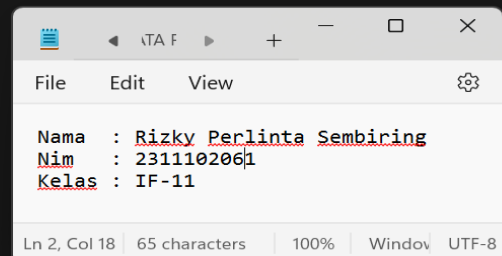
Descendant dari node E adalah: G

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

Deskripsi Program: Program ini mengimplementasikan sebuah pohon yang berjenis pohon biner dengan berbagai fungsi untuk membuat, memanipulasi, dan mengelola pohon. Fungsi-fungsi tersebut meliputi inisialisasi pohon, penambahan node ke kiri atau kanan, pengubahan dan pengambilan data node, serta pencarian informasi node. Selain itu, program ini juga menyediakan metode untuk melakukan penelusuran pre-order, in-order, dan post-order, serta fitur untuk menghitung ukuran dan tinggi pohon. Fungsi tambahan seperti menampilkan child dan descendant dari sebuah node juga disertakan. Program ini memungkinkan pengguna untuk menghapus subtree atau seluruh pohon, serta menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node per level.

D. Kesimpulan

Graph dan tree adalah dua struktur data fundamental dalam C++ yang memiliki peran penting dalam memecahkan masalah komputasi yang kompleks. Graph cocok untuk memodelkan hubungan dan koneksi antara entitas, sementara tree lebih efektif untuk merepresentasikan data yang memiliki struktur hierarkis. Pemahaman dan implementasi yang tepat dari kedua struktur data ini memungkinkan pengembangan solusi yang efisien dan efektif untuk berbagai jenis masalah.

E. Referensi

[1] Asisten Pratikum “Modul 9 Graph dan Tree”, Learning Management System, 2024.

[2] Saputri S, “Graf dan Pohon”. (2016), diakses pada 10 juni , dari

<https://ahmadhadari77.blogspot.com/2019/05/graph-graf-dan-tree-pohon-algoritma.html>

[3] Sruthy, “Implementasi Grafik Dalam C++ Menggunakan Adjacency List”, (2024, 7 maret), diakses pada 10 juni, dari

<https://www.softwaretestinghelp.com/graph-implementation-cpp/>