

LAPORAN PRAKTIKUM

MODUL IX GRAPH DAN TREE



Disusun oleh:
Ahmadan Syaridin

2311102038

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

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023**

BAB I

TUJUAN PRAKTIKUM

- a. Mampu memahami graph dan tree pada struktur data dan algoritma
- b. Mampu mengimplementasikan operasi-operasi pada graph dan tree
- c. Mampu memecahkan permasalahan dengan Solusi graph dan tree

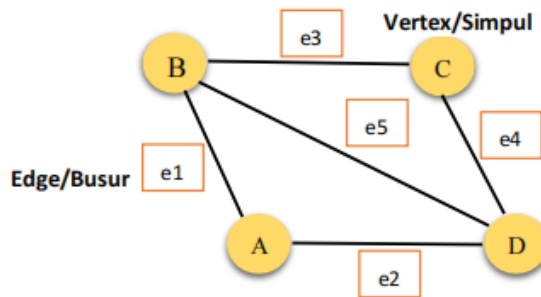
BAB II DASAR TEORI

1. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

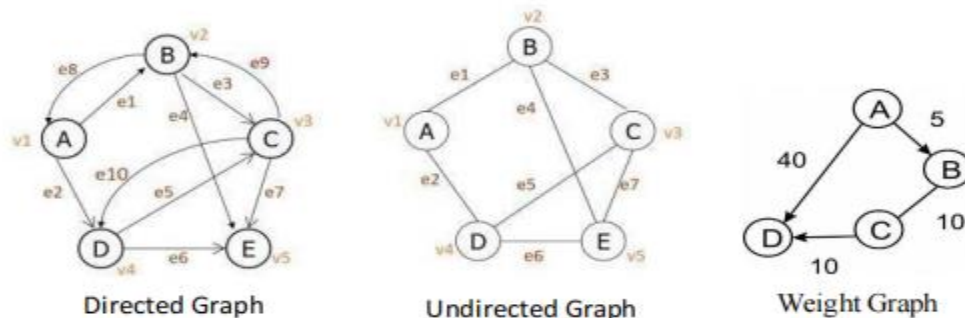
Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde. Dapat digambarkan:



Gambar 1 Contoh Graph

Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

Jenis-jenis Graph

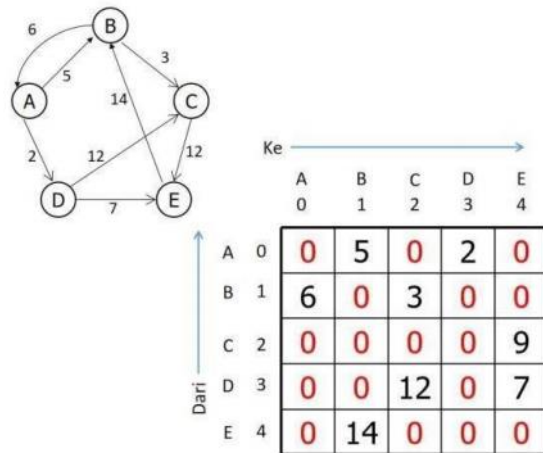


a. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.

b. Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.

c. Weigh Graph : Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks

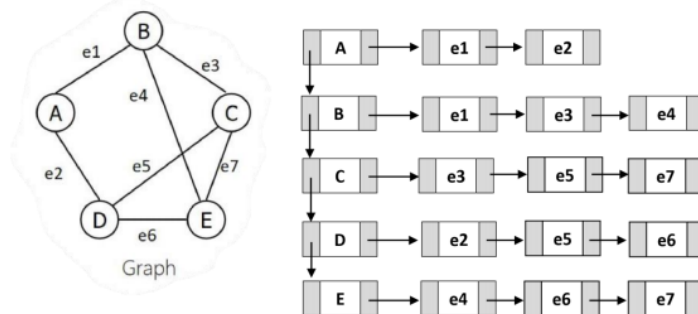


Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

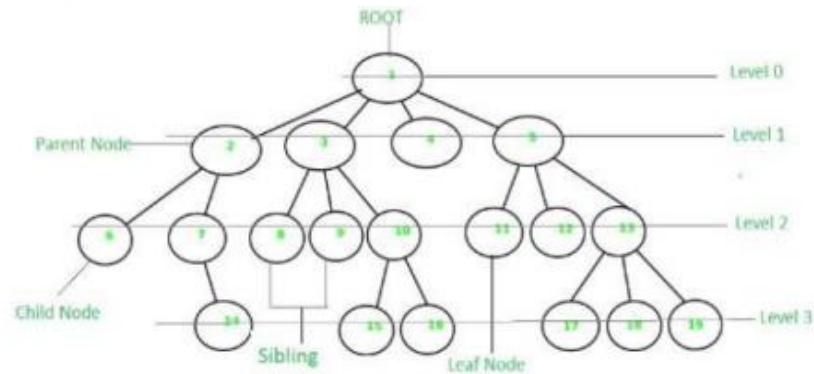
Yang perlu diperhatikan dalam membuat representasi graph dalam bentuk linked list adalah membedakan antara simpul vertex dengan simpul edge. Simpul vertex menyatakan simpul atau vertex dan simpul edge menyatakan busur (hubungan antar simbol). Struktur keduanya bisa sama bisa juga berbeda tergantung kebutuhan, namun biasanya disamakan. Yang membedakan antara simpul vertex dengan simpul edge nantinya adalah anggapan terhadap simpul tersebut juga fungsinya masing-masing.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

Dalam ilmu komputer, pohon adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, di mana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :

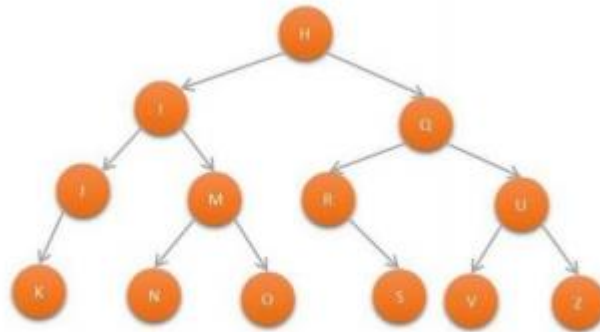


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyarkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.

Gambar 1, menunjukkan contoh dari struktur data binary tree.

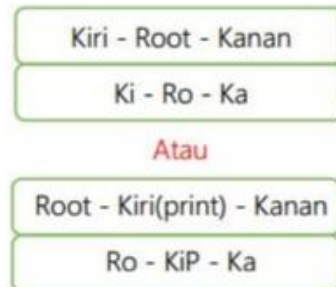


Gambar 1. Struktur Data Binary Tree

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list

3. In-order

- Secara rekursif mencetak seluruh data pada subpohon kiri
 - Cetak data pada root
 - Secara rekursif mencetak seluruh data pada subpohon kanan
- Dapat kita turunkan rumus penelusuran menjadi:

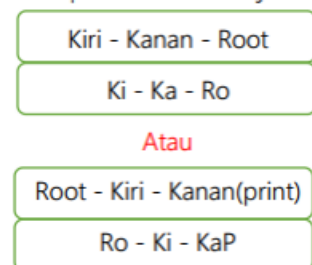


4. Post order

Penelusuran secara in-order memiliki alur:

- Secara rekursif mencetak seluruh data pada subpohon kiri
- Secara rekursif mencetak seluruh data pada subpohon kanan
- Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



BAB III

GUIDED

1. SOURCE CODE PROGRAM GARPH

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogjakarta"};

int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

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

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

```
}  
    return 0;  
}
```

SCREENSHOT

```
PS C:\semester 2\strukdat\tubes-strukdat> & 'c:\Users\ACER\.vscode\extensions\ms-vscode.cp  
WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-djbcmpuk.xue' '--stdout=Microsoft-  
soft-MIEngine-Error-vomwamj1.sh1' '--pid=Microsoft-MIEngine-Pid-n011clfj.kc0' '--dbgExe=C:  
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 C:\semester 2\strukdat\tubes-strukdat> |
```

DESKRIPSI

Program ini adalah program C++ yang mendefinisikan sebuah graph sederhana yang merepresentasikan koneksi antar kota di Jawa Barat dan Jawa Tengah menggunakan array dua dimensi. berisi nama-nama kota, yaitu "Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto", dan "Yogyakarta". Matriks dua dimensi yang merepresentasikan jarak antar kota. Nilai selain nol dalam matriks menunjukkan ada koneksi langsung antara dua kota dengan bobot tertentu.

2. SOURCE CODE PROGRAM TREE

```
3. #include <iostream>  
4. using namespace std;  
5. /// PROGRAM BINARY TREE  
6. // Deklarasi Pohon  
7. struct Pohon  
8. {  
9.     char data;  
10.    Pohon *left, *right, *parent;  
11.};  
12.Pohon *root, *baru;  
13.// Inisialisasi  
14.void init()  
15.{  
16.    root = NULL;  
17.}  
18.// Cek Node  
19.int isEmpty()  
20.{
```



```

21.     if (root == NULL)
22.         return 1;
23.     else
24.         return 0;
25.     // true
26.     // false
27.}
28.// Buat Node Baru
29.void buatNode(char data)
30.{
31.    if (isEmpty() == 1)
32.    {
33.        root = new Pohon();
34.        root->data = data;
35.        root->left = NULL;
36.        root->right = NULL;
37.        root->parent = NULL;
38.        cout << "\n Node " << data << " berhasil dibuat menjadi
root."
39.            << endl;
40.    }
41.    else
42.    {
43.        cout << "\n Pohon sudah dibuat" << endl;
44.    }
45.}
46.// Tambah Kiri
47.Pohon *insertLeft(char data, Pohon *node)
48.{
49.    if (isEmpty() == 1)
50.    {
51.        cout << "\n Buat tree terlebih dahulu!" << endl;
52.        return NULL;
53.    }
54.    else
55.    {
56.        // cek apakah child kiri ada atau tidak
57.        if (node->left != NULL)
58.        {
59.            // kalau ada
60.            cout << "\n Node " << node->data << " sudah ada child
kiri!"
61.                << endl;
62.            return NULL;
63.        }

```

```

64.         else
65.         {
66.             // kalau tidak ada
67.             baru = new Pohon();
68.             baru->data = data;
69.             baru->left = NULL;
70.             baru->right = NULL;
71.             baru->parent = node;
72.             node->left = baru;
73.             cout << "\n Node " << data << " berhasil ditambahkan ke
child kiri " << baru->parent->data << endl;
74.             return baru;
75.         }
76.     }
77.}
78.// Tambah Kanan
79.Pohon *insertRight(char data, Pohon *node)
80.{
81.    if (root == NULL)
82.    {
83.        cout << "\n Buat tree terlebih dahulu!" << endl;
84.        return NULL;
85.    }
86.    else
87.    {
88.        // cek apakah child kanan ada atau tidak
89.        if (node->right != NULL)
90.        {
91.            // kalau ada
92.            cout << "\n Node " << node->data << " sudah ada child
kanan!"
93.                << endl;
94.            return NULL;
95.        }
96.        else
97.        {
98.            // kalau tidak ada
99.            baru = new Pohon();
100.            baru->data = data;
101.            baru->left = NULL;
102.            baru->right = NULL;
103.            baru->parent = node;
104.            node->right = baru;
105.            cout << "\n Node " << data << " berhasil ditambahkan
ke child kanan " << baru->parent->data << endl;

```

```

106.         return baru;
107.     }
108. }
109. }
110. // Ubah Data Tree
111. void update(char data, Pohon *node)
112. {
113.     if (isEmpty() == 1)
114.     {
115.         cout << "\n Buat tree terlebih dahulu!" << endl;
116.     }
117.     else
118.     {
119.         if (!node)
120.             cout << "\n Node yang ingin diganti tidak ada!!" <<
endl;
121.         else
122.         {
123.             char temp = node->data;
124.             node->data = data;
125.             cout << "\n Node " << temp << " berhasil diubah
menjadi " << data << endl;
126.         }
127.     }
128. }
129. // Lihat Isi Data Tree
130. void retrieve(Pohon *node)
131. {
132.     if (!root)
133.     {
134.         cout << "\n Buat tree terlebih dahulu!" << endl;
135.     }
136.     else
137.     {
138.         if (!node)
139.             cout << "\n Node yang ditunjuk tidak ada!" << endl;
140.         else
141.         {
142.             cout << "\n Data node : " << node->data << endl;
143.         }
144.     }
145. }
146. // Cari Data Tree
147. void find(Pohon *node)
148. {

```

```

149.     if (!root)
150.     {
151.         cout << "\n Buat tree terlebih dahulu!" << endl;
152.     }
153.     else
154.     {
155.         if (!node)
156.             cout << "\n Node yang ditunjuk tidak ada!" << endl;
157.         else
158.         {
159.             cout << "\n Data Node : " << node->data << endl;
160.             cout << " Root : " << root->data << endl;
161.             if (!node->parent)
162.                 cout << " Parent : (tidak punya parent)" << endl;
163.             else
164.                 cout << " Parent : " << node->parent->data <<
endl;
165.             if (node->parent != NULL && node->parent->left != node
&&
166.                 node->parent->right == node)
167.                 cout << " Sibling : " << node->parent->left->data
<< endl;
168.             else if (node->parent != NULL && node->parent->right
!= node &&
169.                 node->parent->left == node)
170.                 cout << " Sibling : " << node->parent->right->data
<< endl;
171.             else
172.                 cout << " Sibling : (tidak punya sibling)" <<
endl;
173.             if (!node->left)
174.                 cout << " Child Kiri : (tidak punya Child kiri)"
<< endl;
175.             else
176.                 cout << " Child Kiri : " << node->left->data <<
endl;
177.             if (!node->right)
178.                 cout << " Child Kanan : (tidak punya Child kanan)"
<< endl;
179.             else
180.                 cout << " Child Kanan : " << node->right->data <<
endl;
181.         }
182.     }
183. }

```

```

184. // Penelusuran (Traversal)
185. // preOrder
186. void preOrder(Pohon *node = root)
187. {
188.     if (!root)
189.         cout << "\n Buat tree terlebih dahulu!" << endl;
190.     else
191.     {
192.         if (node != NULL)
193.         {
194.             cout << " " << node->data << ", ";
195.             preOrder(node->left);
196.             preOrder(node->right);
197.         }
198.     }
199. }
200. // inOrder
201. void inOrder(Pohon *node = root)
202. {
203.     if (!root)
204.         cout << "\n Buat tree terlebih dahulu!" << endl;
205.     else
206.     {
207.         if (node != NULL)
208.         {
209.             inOrder(node->left);
210.             cout << " " << node->data << ", ";
211.             inOrder(node->right);
212.         }
213.     }
214. }
215. // postOrder
216. void postOrder(Pohon *node = root)
217. {
218.     if (!root)
219.         cout << "\n Buat tree terlebih dahulu!" << endl;
220.     else
221.     {
222.         if (node != NULL)
223.         {
224.             postOrder(node->left);
225.             postOrder(node->right);
226.             cout << " " << node->data << ", ";
227.         }
228.     }

```

```

229. }
230. // Hapus Node Tree
231. void deleteTree(Pohon *node)
232. {
233.     if (!root)
234.         cout << "\n Buat tree terlebih dahulu!" << endl;
235.     else
236.     {
237.         if (node != NULL)
238.         {
239.             if (node != root)
240.             {
241.                 node->parent->left = NULL;
242.                 node->parent->right = NULL;
243.             }
244.             deleteTree(node->left);
245.             deleteTree(node->right);
246.             if (node == root)
247.             {
248.                 delete root;
249.                 root = NULL;
250.             }
251.             else
252.             {
253.                 delete node;
254.             }
255.         }
256.     }
257. }
258. // Hapus SubTree
259. void deleteSub(Pohon *node)
260. {
261.     if (!root)
262.         cout << "\n Buat tree terlebih dahulu!" << endl;
263.     else
264.     {
265.         deleteTree(node->left);
266.         deleteTree(node->right);
267.         cout << "\n Node subtree " << node->data << " berhasil
           dihapus." << endl;
268.     }
269. }
270. // Hapus Tree
271. void clear()
272. {

```

```

273.     if (!root)
274.         cout << "\n Buat tree terlebih dahulu!!" << endl;
275.     else
276.     {
277.         deleteTree(root);
278.         cout << "\n Pohon berhasil dihapus." << endl;
279.     }
280. }
281. // Cek Size Tree
282. int size(Pohon *node = root)
283. {
284.     if (!root)
285.     {
286.         cout << "\n Buat tree terlebih dahulu!!" << endl;
287.         return 0;
288.     }
289.     else
290.     {
291.         if (!node)
292.         {
293.             return 0;
294.         }
295.         else
296.         {
297.             return 1 + size(node->left) + size(node->right);
298.         }
299.     }
300. }
301. // Cek Height Level Tree
302. int height(Pohon *node = root)
303. {
304.     if (!root)
305.     {
306.         cout << "\n Buat tree terlebih dahulu!" << endl;
307.         return 0;
308.     }
309.     else
310.     {
311.         if (!node)
312.         {
313.             return 0;
314.         }
315.         else
316.         {
317.             int heightKiri = height(node->left);

```

```

318.         int heightKanan = height(node->right);
319.         if (heightKiri >= heightKanan)
320.         {
321.             return heightKiri + 1;
322.         }
323.         else
324.         {
325.             return heightKanan + 1;
326.         }
327.     }
328. }
329. }
330. // Karakteristik Tree
331. void charateristic()
332. {
333.     cout << "\n Size Tree : " << size() << endl;
334.     cout << " Height Tree : " << height() << endl;
335.     cout << " Average Node of Tree : " << size() / height() <<
        endl;
336. }
337. int main()
338. {
339.     buatNode('A');
340.     Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG,
        *nodeH, *nodeI, *nodeJ;
341.     nodeB = insertLeft('B', root);
342.     nodeC = insertRight('C', root);
343.     nodeD = insertLeft('D', nodeB);
344.     nodeE = insertRight('E', nodeB);
345.     nodeF = insertLeft('F', nodeC);
346.     nodeG = insertLeft('G', nodeE);
347.     nodeH = insertRight('H', nodeE);
348.     nodeI = insertLeft('I', nodeG);
349.     nodeJ = insertRight('J', nodeG);
350.     update('Z', nodeC);
351.     update('C', nodeC);
352.     retrieve(nodeC);
353.     find(nodeC);
354.     cout << "\n PreOrder : " << endl;
355.     preOrder(root);
356.     cout << "\n"
357.         << endl;
358.     cout << " InOrder : " << endl;
359.     inOrder(root);
360.     cout << "\n"

```



```
361.         << endl;
362.     cout << " PostOrder :" << endl;
363.     postOrder(root);
364.     cout << "\n"
365.         << endl;
366.     charateristic();
367.     deleteSub(nodeE);
368.     cout << "\n PreOrder :" << endl;
369.     preOrder();
370.     cout << "\n"
371.         << endl;
372.     charateristic();
373. }
374.
```

SCREENSHOT

```
PS C:\semester 2\strukdat\tubes-strukdat> & 'c:\Users\ACER\.vscode\extensions\ms-vscode.c
WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-u0pm5131.kbr' '--stdout=Microsoft
soft-MIEngine-Error-f2x0kxmw.csn' '--pid=Microsoft-MIEngine-Pid-0zckig3v.cz5' '--dbgExe=C:

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

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

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

Node subtree E berhasil dihapus.

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

Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS C:\semester 2\strukdat\tubes-strukdat> █
```

DESKRIPSI PROGRAM

Program ini adalah implementasi dasar dari binary tree (pohon biner) dalam bahasa C++. Mendefinisikan struktur dari node pohon biner dengan data berupa karakter, serta pointer ke left (anak kiri), right (anak kanan), dan parent (orang tua). Output dari program ini akan menunjukkan hasil dari berbagai operasi pada pohon biner, termasuk penambahan node, pengubahan data, penelusuran, dan penghapusan subtree.

BAB IV UNGUIDED

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

SOURCE CODE

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

int main() {
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah simpul : ";
    cin >> jumlah_simpul;

    vector<string>AhmadanSyaridin_2311102038(jumlah_simpul);
    vector<vector<int>> bobot(jumlah_simpul,
vector<int>(jumlah_simpul));

    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << "Silahkan masukkan nama simpul " << i + 1 << " : ";
        cin >> AhmadanSyaridin_2311102038[i];
    }

    cout << "Silahkan masukkan bobot antar simpul\n";

    for (int i = 0; i < jumlah_simpul; ++i) {
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << AhmadanSyaridin_2311102038[i] << "-->" <<
AhmadanSyaridin_2311102038[j] << " : ";
            cin >> bobot[i][j];
        }
    }
    cout << "\n\t";
    for (int i = 0; i < jumlah_simpul; ++i) {
        cout << AhmadanSyaridin_2311102038[i] << "\t";
    }
    cout << "\n";

    for (int i = 0; i < jumlah_simpul; ++i) {
```

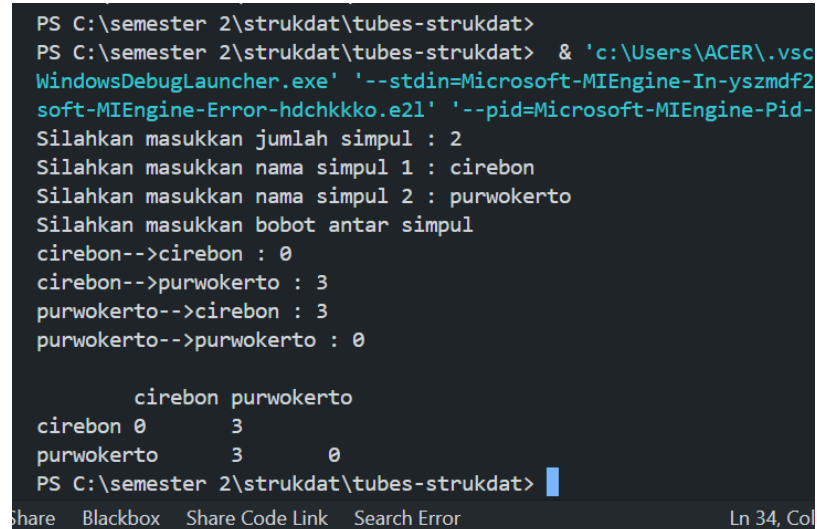
```

        cout << AhmadanSyaridin_2311102038[i] << "\t";
        for (int j = 0; j < jumlah_simpul; ++j) {
            cout << bobot[i][j] << "\t";
        }
        cout << "\n";
    }

    return 0;
}

```

SCREENSHOT



```

PS C:\semester 2\strukdat\tubes-strukdat>
PS C:\semester 2\strukdat\tubes-strukdat> & 'c:\Users\ACER\.vscode\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-yszmdf2soft-MIEngine-Error-hdchkkko.e21' '--pid=Microsoft-MIEngine-Pid-
Silahkan masukkan jumlah simpul : 2
Silahkan masukkan nama simpul 1 : cirebon
Silahkan masukkan nama simpul 2 : purwokerto
Silahkan masukkan bobot antar simpul
cirebon-->cirebon : 0
cirebon-->purwokerto : 3
purwokerto-->cirebon : 3
purwokerto-->purwokerto : 0

      cirebon purwokerto
cirebon 0      3
purwokerto 3      0
PS C:\semester 2\strukdat\tubes-strukdat>

```

DESKRIPSI

Program ini digunakan untuk membuat dan mengelola sebuah graf berbobot. Program ini meminta pengguna untuk memasukkan jumlah simpul (nodes), nama setiap simpul, serta bobot (weight) antar simpul. Hasil akhirnya adalah menampilkan matriks bobot antar simpul.

2. Modifikasi unguided tree diatas dengan program menu menggunakan input data tree dari user!

SOURCE COUDE

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

// Declaring the Tree structure
struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root = nullptr;

// Initialize the tree
void init() {
    root = NULL;
}

// Check if the tree is empty
int isEmpty() {
    return (root == NULL) ? 1 : 0;
}

// Create a new node
Pohon* buatNode(char data) {
    Pohon* newNode = new Pohon();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    newNode->parent = NULL;
    // cout << "\nNode " << data << " berhasil dibuat." << endl;
    return newNode;
}

// Insert a node to the left
Pohon* insertLeft(Pohon* parent, Pohon* child) {
    if (isEmpty() == 1) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->left != NULL) {
```

```

        cout << "\nNode " << parent->left->data << " sudah ada child
kiri!" << endl;
        return NULL;
    } else {
        child->parent = parent;
        parent->left = child;
        // cout << "\nNode " << child->data << " berhasil ditambahkan ke
child kiri " << child->parent->data << endl;
        return child;
    }
}
}

// Insert a node to the right
Pohon* insertRight(Pohon* parent, Pohon* child) {
    if (root == NULL) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (parent->right != NULL) {
            cout << "\nNode " << parent->right->data << " sudah ada child
kanan!" << endl;
            return NULL;
        } else {
            child->parent = parent;
            parent->right = child;
            // cout << "\nNode " << child->data << " berhasil ditambahkan ke
child kanan " << child->parent->data << endl;
            return child;
        }
    }
}

// Update node data
void update(char data, Pohon *node) {
    if (isEmpty() == 1) {
        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;

```

```

    }
}

// Retrieve node data
void retrieve(Pohon *node) {
    if (!root) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

// Find node and display its properties
void find(Pohon *node) {
    if (!root) {
        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;
    }
}

// Pre-order traversal
void preOrder(Pohon *node) {
    if (node != NULL) {
        cout << " " << node->data << ", ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

// In-order traversal
void inOrder(Pohon *node) {
    if (node != NULL) {
        inOrder(node->left);
        cout << " " << node->data << ", ";
        inOrder(node->right);
    }
}

// Post-order traversal
void postOrder(Pohon *node) {
    if (node != NULL) {
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}

// Delete the entire tree
void deleteTree(Pohon *node) {
    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;
        }
    }
}

// Delete a subtree
void deleteSub(Pohon *node) {
    if (!root)
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil dihapus." <<
endl;
    }
}

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

// Get the size of the tree
int size(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        return 1 + size(node->left) + size(node->right);
    }
}

// Get the height of the tree
int height(Pohon *node) {
    if (node == NULL) {
        return 0;
    } else {
        int heightKiri = height(node->left);
        int heightKanan = height(node->right);
        return (heightKiri >= heightKanan) ? heightKiri + 1 : heightKanan + 1;
    }
}

```

```

    }
}

// Display tree characteristics
void characteristic() {
    cout << "\nSize Tree : " << size(root) << endl;
    cout << "Height Tree : " << height(root) << endl;
    cout << "Average Node of Tree : " << (size(root) / (float)height(root)) <<
endl;
}

int main() {
    root = buatNode('A');
    int menu, part, part2;
    char zefanyabranatertiustarigan_2311102028;

    vector<Pohon*> nodes;
    nodes.push_back(buatNode('B'));
    nodes.push_back(buatNode('C'));
    nodes.push_back(buatNode('D'));
    nodes.push_back(buatNode('E'));
    nodes.push_back(buatNode('F'));
    nodes.push_back(buatNode('G'));
    nodes.push_back(buatNode('H'));
    nodes.push_back(buatNode('I'));
    nodes.push_back(buatNode('J'));

    insertLeft(root, nodes[0]);
    insertRight(root, nodes[1]);
    insertLeft(nodes[0], nodes[2]);
    insertRight(nodes[0], nodes[3]);
    insertLeft(nodes[1], nodes[4]);
    insertLeft(nodes[3], nodes[5]);
    insertRight(nodes[3], nodes[6]);
    insertLeft(nodes[5], nodes[7]);
    insertRight(nodes[5], nodes[8]);

    do
    {
        cout << "\n----- PROGHRAM GRAPH ----- \n"
        "1. Tambah node\n"
        "2. Tambah di kiri\n"
        "3. Tambah di kanan\n"
        "4. Lihat karakteristik tree\n"
        "5. Lihat isi data tree\n"

```

```

"6. Cari data tree\n"
"7. Penelusuran (Traversal) preOrder\n"
"8. Penelusuran (Traversal) inOrder\n"
"9. Penelusuran (Traversal) postOrder\n"
"10. Hapus subTree\n"
"0. KELUAR\n"
"\nPilih : ";
cin >> menu;
cout << "-----Running Command...\n";
switch (menu) {
    case 1:
        cout << "\n Nama Node (Character) : ";
        cin >> zefanyabranatertiustarigan_2311102028;
        nodes.push_back(buatNode(zefanyabranatertiustarigan_2311102028
));
        break;
    case 2:
        cout << "\nMasukkan nomor untuk node parent : ";
        cin >> part;
        cout << "\nMasukkan nomor untuk node child : ";
        cin >> part2;
        insertLeft(nodes[part], nodes[part2]);
        break;
    case 3:
        cout << "\nMasukkan nomor untuk node parent : ";
        cin >> part;
        cout << "\nMasukkan nomor untuk node child : ";
        cin >> part2;
        insertRight(nodes[part], nodes[part2]);
        break;
    case 4:
        charateristic();
        break;
    case 5:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        retrieve(nodes[part]);
        break;
    case 6:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        find(nodes[part]);
        break;
    case 7:
        cout << "\nPreOrder : " << endl;

```

```

        preOrder(root);
        cout << "\n" << endl;
        break;
    case 8:
        cout << "\nInOrder :" << endl;
        inOrder(root);
        cout << "\n" << endl;
        break;
    case 9:
        cout << "\nPostOrder :" << endl;
        postOrder(root);
        cout << "\n" << endl;
        break;
    case 10:
        cout << "\nMasukkan nomor node : ";
        cin >> part;
        deleteSub(nodes[part]);
        break;
    default:
        break;
}
} while (menu != 0);
}

```

SCREENSHOT

```

PS C:\semester 2\strukdat\tubes-strukdat> & 'c:\Users\ACER\.vscode
WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-qt150fpr.m
soft-MIEngine-Error-3lyc3y2z.bzl' '--pid=Microsoft-MIEngine-Pid-nwa

----- PROGHRAH GRAPH -----
1. Tambah node
2. Tambah di kiri
3. Tambah di kanan
4. Lihat karakteristik tree
5. Lihat isi data tree
6. Cari data tree
7. Penelurusan (Traversal) preOrder
8. Penelurusan (Traversal) inOrder
9. Penelurusan (Traversal) postOrder
10. Hapus subTree
0. KELUAR

Pilih : 4
-----Running Command...

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

```

DESKRIPSI PROGRAM

Program ini menampilkan menu interaktif yang memungkinkan pengguna untuk memilih berbagai operasi pada pohon biner. Pengguna dapat menambah node, menambah anak kiri atau kanan, melihat karakteristik pohon, menelusuri pohon dengan metode traversal, dan menghapus subtree.

BAB V

KESIMPULAN

Modul 9 mengenai "Graph dan Tree" bertujuan untuk membantu mahasiswa memahami konsep dan penerapan dua struktur data esensial dalam ilmu komputer: graph dan tree. Graph menggambarkan hubungan antara objek melalui node (simpul) dan edge (busur), serta dapat digunakan dalam berbagai aplikasi seperti jaringan sosial dan pemetaan jalan. Terdapat beberapa jenis graph, termasuk graph berarah, graph tak berarah, dan weight graph, yang dapat direpresentasikan dengan matriks atau linked list. Sementara itu, tree adalah struktur data hierarkis dengan node yang memiliki hubungan induk-anak, sering digunakan untuk menyimpan data terstruktur seperti silsilah keluarga atau struktur organisasi. Modul ini juga menjelaskan operasi dasar pada tree, seperti pembuatan, penghapusan, pencarian, dan penelusuran (traversal) dengan metode pre-order, in-order, dan post-order, serta memberikan panduan praktis untuk mengimplementasikan graph dan tree dalam pemrograman menggunakan bahasa C++.

DAFTAR ISI

1. Modul 2 Grap dan Tree
2. Weiss, M. A. (2013). Data Structures and Algorithm Analysis in C++. Pearson