LAPORAN PRAKTIKUM

MODUL 9
GRAPH DAN TREE

Disusun oleh:
Denny Budiansyach
NIM: 2311102022

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

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

BAB I
TUJUAN PRAKTIKUM

1. Praktikan diharapkan mampu memahami graph dan tree
2. Praktikan diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II

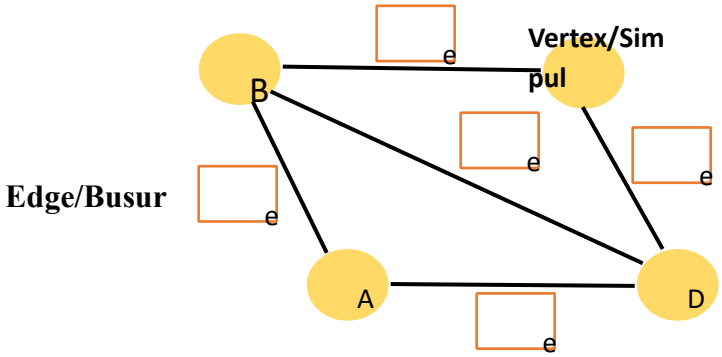
DASAR TEORI

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 sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

G = (V, E)

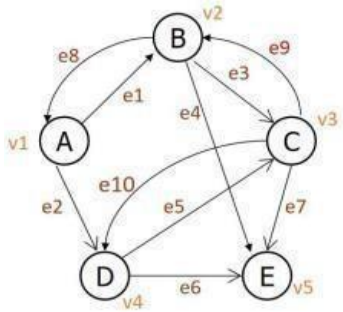
Dimana G adalah Graph, V adalah simpul atau vertex dan E sebagai sisi atau edge. Dapat digambarkan:



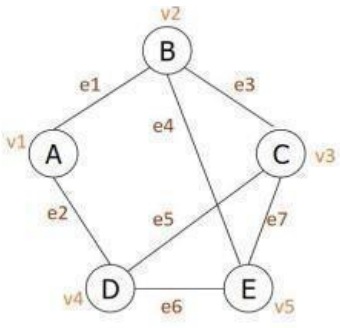
Gambar 1 Contoh Graph

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

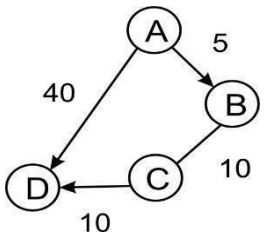
Jenis-jenis Graph



Directed Graph



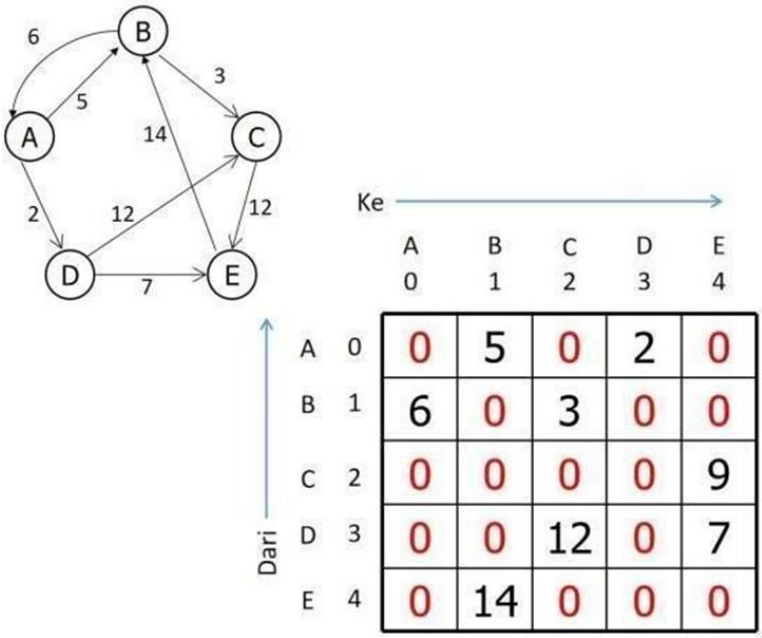
Undirected Graph



Weight Graph

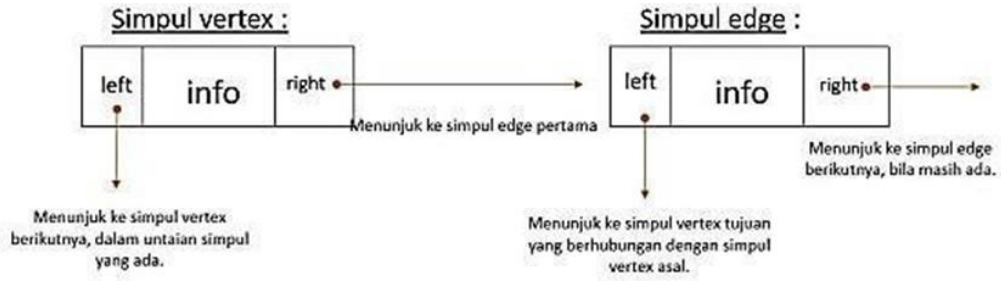
a. **Graph berarah (directed graph):** Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
b. **Graph tak berarah (undirected graph):** Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
c. **Weight Graph** : Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks



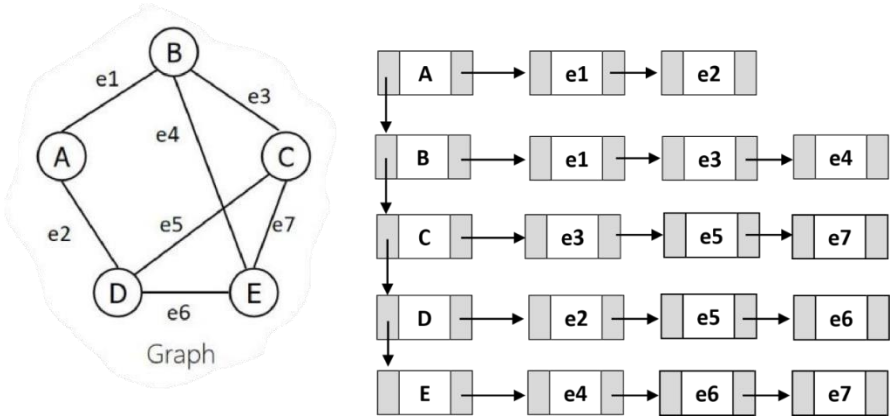
Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List



Gambar 5 Representasi Graph dengan Linked List

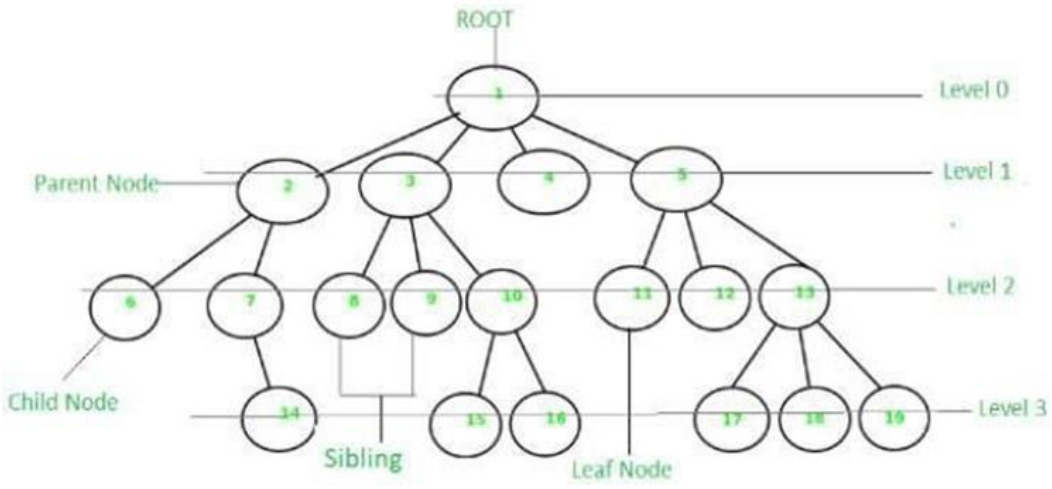
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan menggunakan keduanya dalam representasi graf.



Gambar 6 Representasi Graph dengan Linked List

2. Tree atau Pohon

Dalam ilmu komputer, pohon/tree 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, dimana 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 hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :

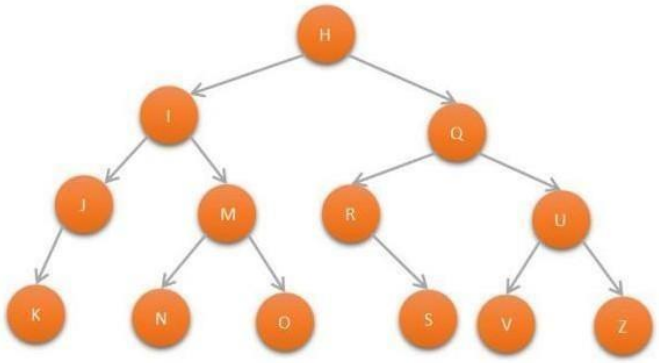


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor 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 diprasyaratkan memiliki simpul satu level di bawahnya (child)

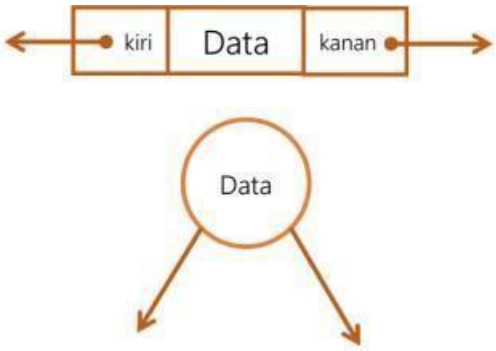
tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



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.

```
struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

a. **Create**: digunakan untuk membentuk binary tree baru yang masih kosong.
b. **Clear**: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
c. **isEmpty**: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.

d. **Insert**: digunakan untuk memasukkan sebuah node kedalam tree.
e. **Find**: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
f. **Update**: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong
g. **Retrive**: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
h. **Delete Sub**: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
i. **Characteristic**: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
j. **Traverse**: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

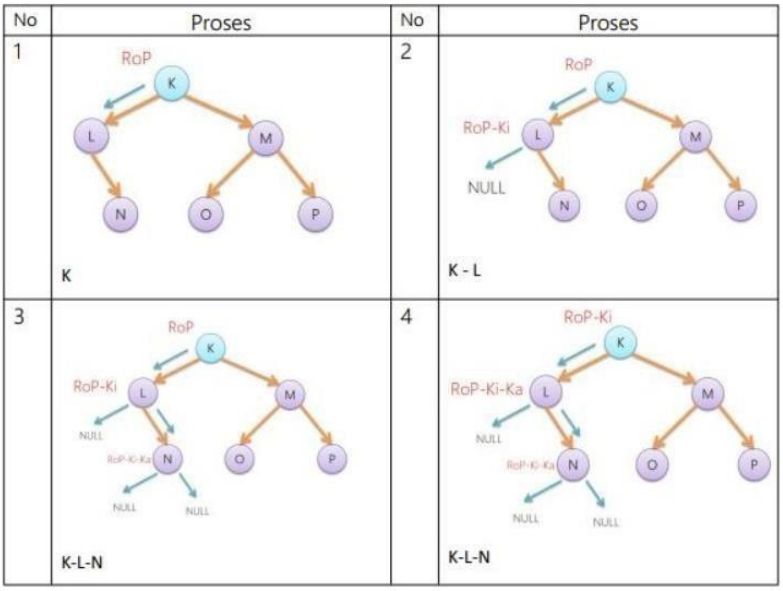
1. Pre-Order

1. Penelusuran secara pre-order memiliki alur: a. Cetak data pada simpul root
2. Secara rekursif mencetak seluruh data pada subpohon kiri
3. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

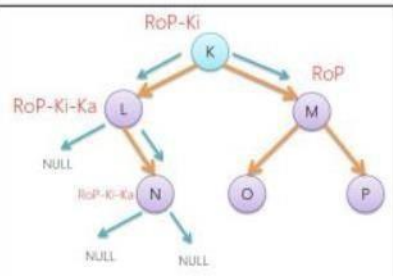
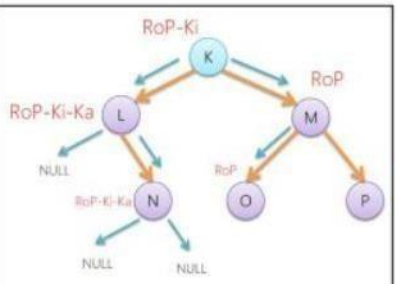
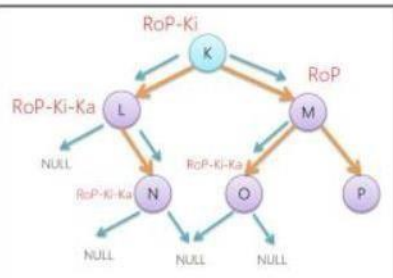
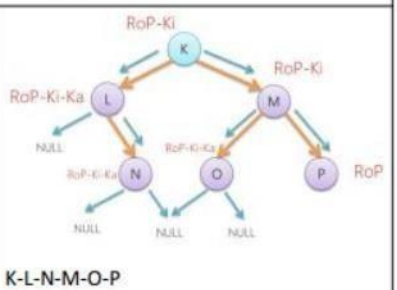
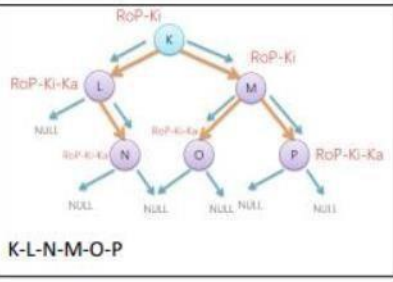
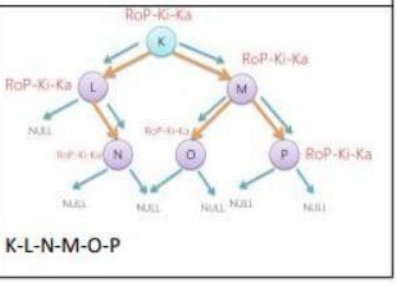
Alur pre-order

Root (print) - Kiri - Kanan

RoP - Ki - Ka



\

5	 K-L-N-M	6	 K-L-N-M-O
7	 K-L-N-M-O	8	 K-L-N-M-O-P
9	 K-L-N-M-O-P	10	 K-L-N-M-O-P

2. In-Order

Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri
b. Cetak data pada root
c. Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

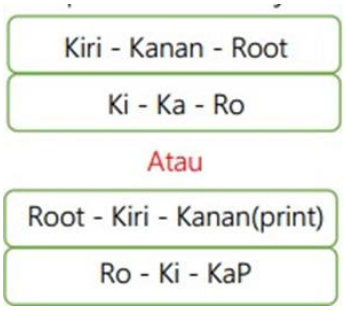
Ro - KiP - Ka

3. Post Order

Penelusuran secara in-order memiliki alur:

a. Secara rekursif mencetak seluruh data pada subpohon kiri
b. Secara rekursif mencetak seluruh data pada subpohon kanan
c. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:



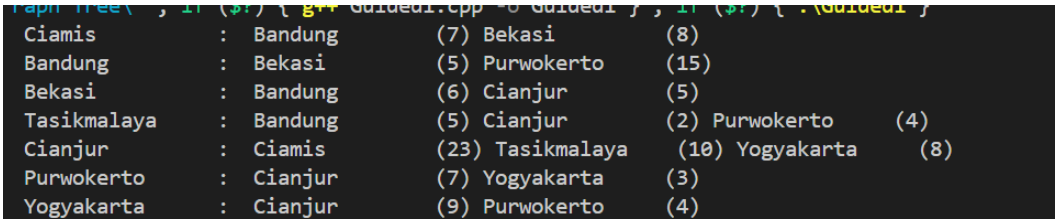
BAB III

GUIDED 1

Guided 1 Source code

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

const string simpul[7] = {
    "Ciamis",
    "Bandung",
    "Bekasi",
    "Tasikmalaya",
    "Cianjur",
    "Purwokerto",
    "Yogyakarta"
};
const 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 << " " << setw(15) << left << simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++) {
            if (busur[baris][kolom] != 0) {
                cout << " " << setw(15) << left << simpul[kolom] <<
"(" << busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}
int main() {
    tampilGraph();
    return 0;
}
```

Screenshoot program



Deskripsi program

Program di atas adalah bentuk implementasi dari konsep representasi graf berarah yang menggunakan matriks bobot untuk menyimpan dan mengelola data jarak antar simpul (node). Program ini merepresentasikan simpul sebagai kota-kota yang disimpan dalam array `simpul` dan jarak antar kota yang disimpan dalam matriks `busur`. Setiap elemen `busur[i][j]` menyimpan jarak dari kota `simpul[i]` ke kota `simpul[j]`, dengan nilai 0 menunjukkan bahwa tidak ada jalur langsung antara kedua kota tersebut. Program ini menggunakan fungsi `tampilGraph()` untuk menampilkan graf, di mana setiap baris menunjukkan kota asal dan daftar kota tujuan beserta jaraknya. Fungsi ini mengiterasi melalui matriks `busur`, dan jika jaraknya bukan 0, ia mencetak nama kota tujuan beserta jaraknya.

GUIDED 2

Guided 2 Source code

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

// pointer global
Pohon *root;

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

bool isEmpty() {
    return root == NULL;
}

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data) {
    if (isEmpty()) {
        root = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi
root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}
```

```
Pohon *insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->left != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kiri " << node->data << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kanan " << node->data << endl;
            return baru;
        }
    }
}

void update(char data, Pohon *node) {
    if (isEmpty()) {
```

```
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" << endl;
        else {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah menjadi "
<< data << endl;
        }
    }
}

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

void find(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    } else {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else {
            cout << "\nData Node : " << node->data << endl;
            cout << "Root : " << root->data << endl;
            if (!node->parent)
                cout << "Parent : (tidak punya parent)" << endl;
            else
                cout << "Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node
&& node->parent->right == node)
                cout << "Sibling : " << node->parent->left->data <<
endl;
```

```
            else if (node->parent != NULL && node->parent->right !=
node && node->parent->left == node)
                cout << "Sibling : " << node->parent->right->data <<
endl;
            else
                cout << "Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << "Child Kiri : (tidak punya Child kiri)" <<
endl;
            else
                cout << "Child Kiri : " << node->left->data << endl;
            if (!node->right)
                cout << "Child Kanan : (tidak punya Child kanan)" <<
endl;
            else
                cout << "Child Kanan : " << node->right->data <<
endl;
        }
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            inOrder(node->left);
```

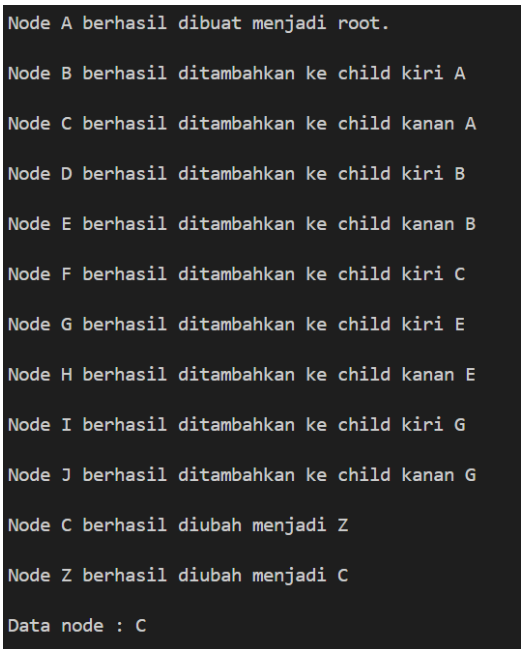
```
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        if (node != NULL) {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}

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

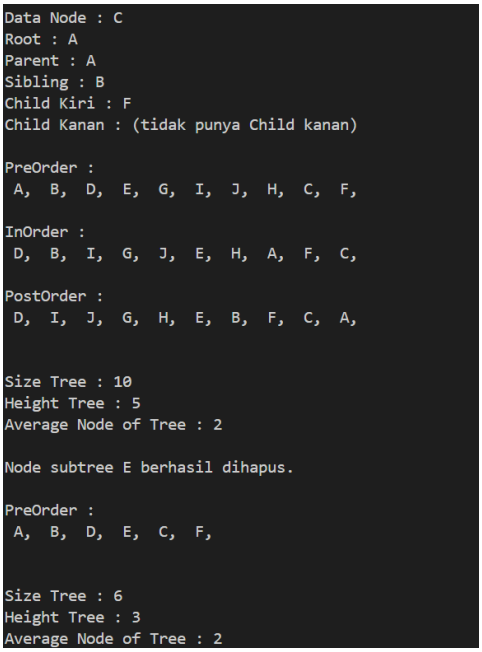
```
// Hapus SubTree
void deleteSub(Pohon *node) {
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

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

// Cek Size Tree
int size(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!!" << endl;
        return 0;
    } else {
        if (!node) {
            return 0;
        } else {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return 0;
```

```
    } else {
        if (!node) {
            return 0;
        } else {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan) {
                return heightKiri + 1;
            } else {
                return heightKanan + 1;
            }
        }
    }
}

// Karakteristik Tree
void characteristic() {
    int s = size(root);
    int h = height(root);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

int main() {
    init();
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
*nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
```

```
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);

    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;

    cout << "InOrder :" << endl;
    inOrder(root);
    cout << "\n" << endl;

    cout << "PostOrder :" << endl;
    postOrder(root);
    cout << "\n" << endl;

    characteristic();

    deleteSub(nodeE);

    cout << "\nPreOrder :" << endl;
    preOrder(root);
    cout << "\n" << endl;

    characteristic();

    return 0;
}
```

Screenshoot program





Deskripsi program

Program di atas mengimplementasikan struktur data pohon biner dengan berbagai fungsi untuk membuat, mengelola, dan memanipulasi pohon tersebut. Program menggunakan struktur `Pohon` untuk merepresentasikan node dengan data karakter dan tiga pointer (parent, left, right).

Fungsi `init()` menginisialisasi pohon sebagai kosong, sedangkan `isEmpty()` mengecek apakah pohon kosong. Fungsi `newPohon(char data)` membuat node baru, dan `buatNode(char data)` menambahkan node tersebut sebagai root jika pohon kosong. Fungsi `insertLeft` dan `insertRight` menambahkan node baru sebagai anak kiri atau kanan dari node yang ditentukan.

Fungsi `update` mengganti data pada node tertentu, sementara `retrieve` menampilkan data node, dan `find` menampilkan informasi lengkap tentang node termasuk parent, sibling, dan anak-anaknya.

Tiga fungsi traversal (`preOrder`, `inOrder`, `postOrder`) berguna untuk menelusuri pohon dan mencetak data node dalam urutan yang berbeda. Fungsi

`deleteTree` dan `deleteSub` menghapus seluruh pohon atau sub-pohon dari node tertentu, sedangkan `clear` menghapus seluruh pohon.

Fungsi `size` menghitung jumlah node dalam pohon, dan `height` mengukur tinggi pohon. Fungsi `characteristic` menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata jumlah node per level.

Dalam fungsi `main`, program membuat pohon, menambahkan dan mengupdate node, serta menampilkan hasil traversal dan karakteristik pohon sebelum dan sesudah penghapusan sub-pohon.

LATIHAN KELAS - UNGUIDED

1. **Unguided Source code**

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

int main() {
    int jumlahSimpul;

    cout << "Silakan masukan jumlah simpul: ";
    cin >> jumlahSimpul;
    cin.ignore();

    vector<string> namaSimpul(jumlahSimpul);

    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << "Simpul " << i + 1 << " : ";
        getline(cin, namaSimpul[i]);
    }

    vector<vector<int>> bobot(jumlahSimpul,
vector<int>(jumlahSimpul, 0));

    cout << "Silakan masukan bobot antar simpul" << endl;
    for (int i = 0; i < jumlahSimpul; ++i) {
        for (int j = 0; j < jumlahSimpul; ++j) {
            if (i != j) {
                cout << namaSimpul[i] << "--->" << namaSimpul[j] <<
" = ";
                cin >> bobot[i][j];
            }
        }
    }

    cout << setw(15) << "";
    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << setw(15) << namaSimpul[i];
```

```
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul; ++i) {
        cout << setw(15) << namaSimpul[i];
        for (int j = 0; j < jumlahSimpul; ++j) {
            cout << setw(15) << bobot[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Screenshoot program

```
Silakan masukan jumlah simpul: 3
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Simpul 3 : MEDAN
Silakan masukan bobot antar simpul
BALI--->PALU = 3
BALI--->MEDAN = 5
PALU--->BALI = 4
PALU--->MEDAN = 3
MEDAN--->BALI = 4
MEDAN--->PALU = 4
                      BALI           PALU          MEDAN
          BALI           0              3              5
          PALU           4              0              3
         MEDAN           4              4              0
```

Deskripsi program

Program di atas adalah bentuk implementasi graph untuk menghitung dan menampilkan jarak antar kota (atau simpul) dalam bentuk matriks dengan menggunakan input dari pengguna. Pertama, program meminta pengguna untuk memasukkan jumlah simpul atau kota. Kemudian, pengguna diminta untuk memasukkan nama setiap simpul.

Setelah nama-nama simpul dimasukkan, program meminta pengguna untuk memasukkan bobot atau jarak antar setiap pasangan simpul yang berbeda. Semua bobot antar simpul disimpan dalam sebuah matriks dua dimensi, dengan nilai awal semua elemen matriks diinisialisasi dengan nol.

Program kemudian mencetak sebuah tabel jarak antar simpul dalam format matriks yang rapi. Nama-nama simpul dicetak sebagai header baris dan kolom, dan bobot antar simpul dicetak di dalam tabel sesuai dengan input yang telah diberikan. Dengan cara ini, pengguna dapat dengan mudah melihat jarak antara setiap pasangan simpul dalam graf yang dihasilkan.

2. **Unguided Source code**

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

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

Pohon *root;

void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

Pohon *newPohon(char data) {
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data) {
    if (isEmpty()) {
        root = newPohon(data);
```

```
        cout << "\nNode " << data << " berhasil dibuat menjadi
root." << endl;
    } else {
        cout << "\nPohon sudah dibuat" << endl;
    }
}

Pohon *insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->left != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kiri!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kiri " << node->data << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    } else {
        if (node->right != NULL) {
            cout << "\nNode " << node->data << " sudah ada child
kanan!" << endl;
            return NULL;
        } else {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kanan " << node->data << endl;
```

```
            return baru;
        }
    }
}

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

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

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

void displayChild(Pohon *node) {
    if (node != NULL) {
        cout << "Parent: " << node->data << endl;
        if (node->left != NULL) {
            cout << "Child Kiri: " << node->left->data << endl;
        } else {
            cout << "Child Kiri: (tidak punya child kiri)" << endl;
        }
        if (node->right != NULL) {
            cout << "Child Kanan: " << node->right->data << endl;
        } else {
            cout << "Child Kanan: (tidak punya child kanan)" <<
endl;
```

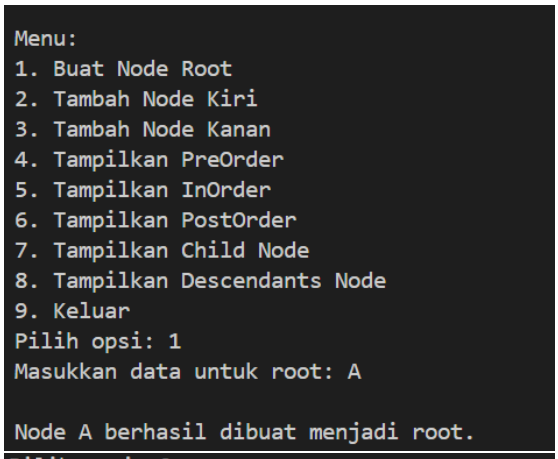
```
        }
    }
}

void displayDescendants(Pohon *node) {
    if (node != NULL) {
        cout << "Descendants of " << node->data << ": ";
        queue<Pohon *> q;
        if (node->left != NULL) q.push(node->left);
        if (node->right != NULL) q.push(node->right);
        while (!q.empty()) {
            Pohon *temp = q.front();
            q.pop();
            cout << temp->data << " ";
            if (temp->left != NULL) q.push(temp->left);
            if (temp->right != NULL) q.push(temp->right);
        }
        cout << endl;
    }
}

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

int main() {
    int choice;
    char data;
    init();

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Buat Node Root\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
        cout << "4. Tampilkan PreOrder\n";
        cout << "5. Tampilkan InOrder\n";
        cout << "6. Tampilkan PostOrder\n";
```

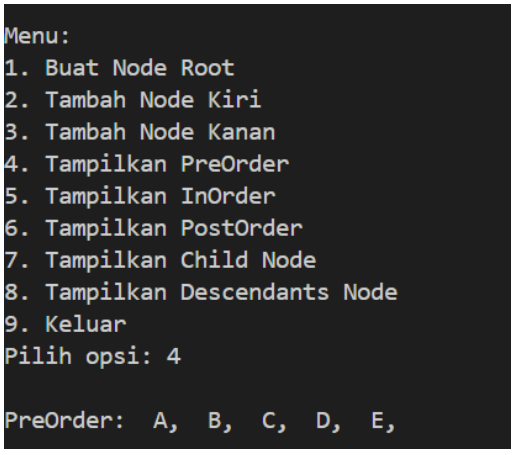
```
        cout << "7. Tampilkan Child Node\n";
        cout << "8. Tampilkan Descendants Node\n";
        cout << "9. Keluar\n";
        cout << "Pilih opsi: ";
        cin >> choice;
        cin.ignore();
        switch (choice) {
            case 1:
                cout << "Masukkan data untuk root: ";
                cin >> data;
                buatNode(data);
                break;
            case 2: {
                cout << "Masukkan data untuk node kiri: ";
                cin >> data;
                cout << "Masukkan data parent: ";
                char parentData;
                cin >> parentData;
                Pohon *parent = findNode(root, parentData);
                if (parent != NULL) {
                    insertLeft(data, parent);
                } else {
                    cout << "\nParent tidak ditemukan!" << endl;
                }
                break;
            }
            case 3: {
                cout << "Masukkan data untuk node kanan: ";
                cin >> data;
                cout << "Masukkan data parent: ";
                char parentData;
                cin >> parentData;
                Pohon *parent = findNode(root, parentData);
                if (parent != NULL) {
                    insertRight(data, parent);
                } else {
                    cout << "\nParent tidak ditemukan!" << endl;
                }
                break;
            }
            case 4:
                cout << "\nPreOrder: ";
```

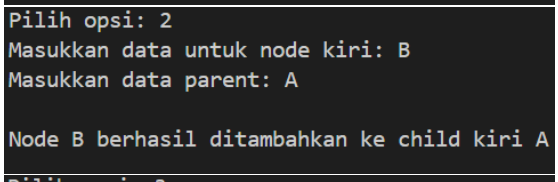
```
                preOrder(root);
                cout << "\n";
                break;
            case 5:
                cout << "\nInOrder: ";
                inOrder(root);
                cout << "\n";
                break;
            case 6:
                cout << "\nPostOrder: ";
                postOrder(root);
                cout << "\n";
                break;
            case 7:
                cout << "Masukkan data node untuk menampilkan child:
";
                cin >> data;
                {
                    Pohon *node = findNode(root, data);
                    if (node != NULL) {
                        displayChild(node);
                    } else {
                        cout << "\nNode tidak ditemukan!" << endl;
                    }
                }
                break;
            case 8:
                cout << "Masukkan data node untuk menampilkan
descendants: ";
                cin >> data;
                {
                    Pohon *node = findNode(root, data);
                    if (node != NULL) {
                        displayDescendants(node);
                    } else {
                        cout << "\nNode tidak ditemukan!" << endl;
                    }
                }
                break;
            case 9:
                return 0;
            default:
```

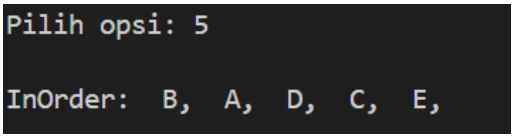
```
                cout << "Opsi tidak valid! Coba lagi." << endl;
            }
        }

        return 0;
}
```

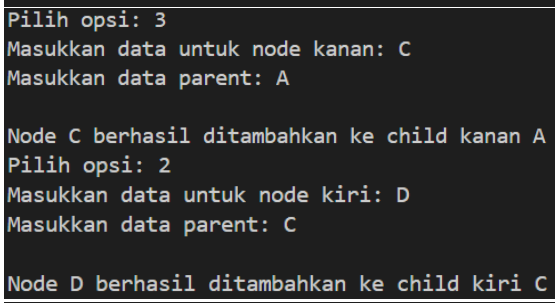
Screenshoot program

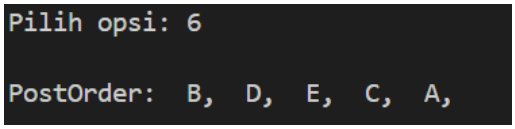


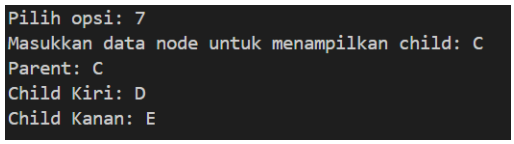


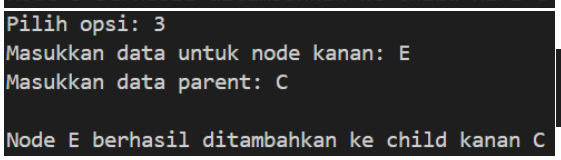


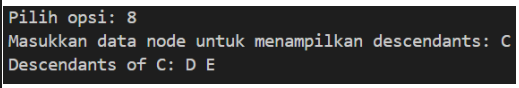












Deskripsi program

Program di atas adalah bentuk implementasi graph dengan beberapa operasi dasar. Pada inisialisasi program, pohon diset pohon kosong. Kemudian user diberi menu untuk dapat membuat root node, menambah anak kiri atau kanan ke node yang sudah ada, dan memeriksa apakah pohon kosong. Terdapat tiga jenis traversal yang disediakan oleh program ini: pre-order, in-order, dan post-order. Selain itu, program memungkinkan user untuk menampilkan anak dari node tertentu dan semua keturunannya. Fungsi pencarian digunakan untuk menemukan node berdasarkan data. Program berjalan dalam loop menu interaktif sampai pengguna memilih untuk keluar.

BAB IV
KESIMPULAN

Graf dan pohon adalah struktur data penting yang digunakan untuk merepresentasikan hubungan antara objek. Graf terdiri dari simpul (vertex) dan sisi (edge) yang bisa berarah (directed), tidak berarah (undirected), atau berbobot (weighted). Penting untuk memahami perbedaan antara simpul vertex dan simpul edge dalam representasi graf.

Pohon, khususnya pohon biner, merupakan jenis graf khusus yang sering digunakan untuk menyimpan data hierarkis. Pohon memiliki simpul induk dan simpul anak, dengan operasi dasar seperti membuat, mengosongkan, memeriksa, menyisipkan, mencari, memperbarui, mengambil, menghapus subtree, mengetahui karakteristik, dan traversal (pre-order, in-order, post-order). Operasi-operasi ini membantu mengelola dan memanipulasi data secara efisien dalam struktur pohon.

DAFTAR PUSTAKA

GeeksforGeeks. (2024, April 3). Diambil kembali dari geeksforgeeks.com: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Asisten Praktikum. (2024). *MODUL 9 – GRAPH DAN TREE*, Learning Managament System.