

LAPORAN PRAKTIKUM

MODUL IX “Graph and Tree“



Disusun oleh:

Shiva Indah Kurnia
NIM 2311102035

Dosen Pengampu:

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

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

TUJUAN PRAKTIKUM

Setelah melakukan praktikum ini diharapkan mahasiswa dapat:

1. Mahasiswa diharapkan mampu menjelaskan dan memahami konsep dari Graph dan Tree itu sendiri.
2. Mahasiswa diharapkan mampu mengaplikasikan Graph dan Tree kedalam pemrograman C++.
3. Mahasiswa diharapkan mampu menyelesaikan studi kasus menggunakan penyelesaian Graph dan Tree.

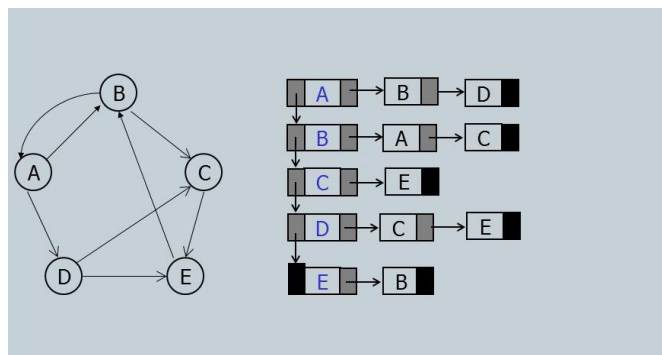
DASAR TEORI

Graph dan tree adalah dua konsep yang berbeda dalam ilmu komputer. Berikut adalah penjelasan singkat mengenai keduanya:

Graph

Graph adalah struktur data yang terdiri dari simpul (node) dan busur (edge) yang menghubungkan simpul-simpul tersebut. Graph digunakan untuk merepresentasikan hubungan antara objek atau entitas dalam suatu sistem. Graph dapat digunakan untuk memodelkan berbagai macam masalah, seperti jaringan sosial, rute perjalanan, dan sebagainya.

Representasi graph dengan linked list adalah salah satu cara merepresentasikan graf dalam bentuk struktur data linked list. Pada representasi ini, setiap simpul pada graf direpresentasikan sebagai node pada linked list, dan setiap busur pada graf direpresentasikan sebagai elemen pada linked list tersebut. Dalam representasi ini, setiap node pada linked list memiliki dua bagian, yaitu data dan pointer ke node berikutnya pada linked list. Data pada setiap node merepresentasikan simpul pada graf, sedangkan pointer pada setiap node merepresentasikan busur pada graf yang terhubung dengan simpul tersebut. Representasi graph dengan linked list dapat digunakan untuk merepresentasikan graf berarah maupun tidak berarah, serta graf berbobot maupun tidak berbobot. Representasi ini memiliki keuntungan dalam penggunaan memori, karena hanya menyimpan busur yang ada pada graf, sehingga lebih efisien untuk graf yang sangat besar. Namun, representasi ini memiliki kelemahan dalam akses data, karena pencarian data pada linked list membutuhkan waktu yang lebih lama dibandingkan dengan representasi graph lainnya seperti adjacency matrix.



Gambar 1. Reprerstasi Graph Pada Linked List

Tree

Tree adalah struktur data khusus dari graph yang tidak memiliki siklus (cycle). Tree terdiri dari simpul-simpul yang terhubung dengan busur-busur, namun tidak ada dua simpul yang terhubung dengan lebih dari satu busur. Tree digunakan untuk merepresentasikan hierarki, seperti struktur direktori pada sistem operasi, atau hubungan antara kelas dalam suatu program.

Operasi pemrograman pada tree meliputi beberapa hal, antara lain:

1. **Traverse:** operasi kunjungan terhadap node-node dalam pohon. Terdapat beberapa jenis traverse, seperti in-order, pre-order, dan post-order traverse. In-order traverse dilakukan dengan cara mengunjungi simpul kiri, mencetak simpul yang dikunjungi, dan mengunjungi simpul kanan. Pre-order traverse dilakukan dengan cara mencetak simpul yang dikunjungi, mengunjungi simpul kiri, dan mengunjungi simpul kanan. Post-order traverse dilakukan dengan cara mengunjungi simpul kiri, mengunjungi simpul kanan, dan mencetak simpul yang dikunjungi.
2. **Insert:** operasi untuk menambahkan simpul baru ke dalam tree. Pada operasi ini, simpul baru ditempatkan pada posisi yang tepat sesuai dengan aturan tree.
3. **Delete:** operasi untuk menghapus simpul dari tree. Pada operasi ini, simpul yang dihapus harus diganti dengan simpul lain yang sesuai dengan aturan tree.
4. **Search:** operasi untuk mencari simpul pada tree. Pada operasi ini, simpul dicari dengan cara menelusuri tree dari simpul root hingga simpul yang dicari ditemukan.
5. **Balance:** operasi untuk menjaga keseimbangan tree. Pada operasi ini, tree diubah sehingga memiliki ketinggian yang seimbang dan meminimalkan waktu akses data.

GUIDED

1. Guided 1

Source code

```
#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}};

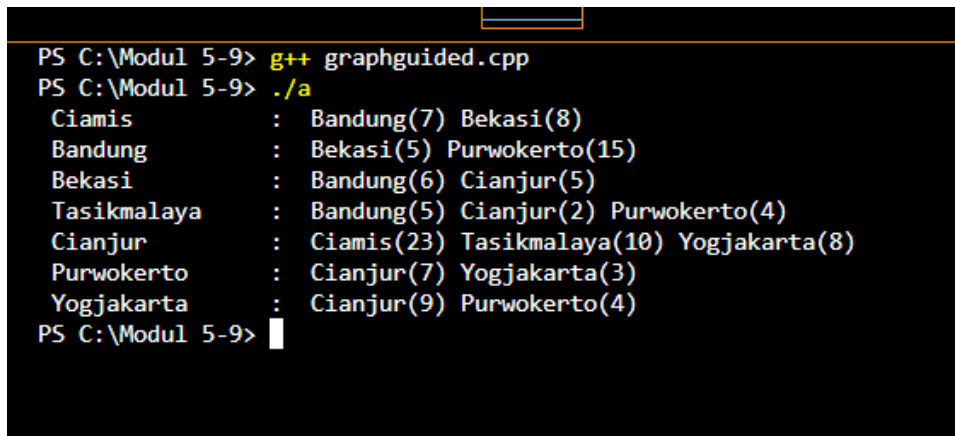
void tampilGraph()
{
    for(int baris =0; baris <7; baris++){
```

```

        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 program



```

PS C:\Modul 5-9> g++ graphguided.cpp
PS C:\Modul 5-9> ./a
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:\Modul 5-9>

```

Deskripsi Program

Program diatas adalah program yang menampilkan graf berbentuk adjacency list. Program ini menggunakan array simpul dan busur untuk merepresentasikan graf. Array simpul berisi nama simpul-simpul pada graf, sedangkan array busur berisi bobot dari setiap busur pada graf. Program ini memiliki satu fungsi yaitu tampilGraph() yang digunakan untuk menampilkan graf dalam bentuk adjacency list. Fungsi ini menggunakan dua perulangan for untuk mengakses setiap elemen pada

array simpul dan busur. Pada setiap iterasi, fungsi ini menampilkan nama simpul dan busur yang terhubung dengan simpul tersebut beserta bobotnya. Program ini kemudian memanggil fungsi tampilGraph() pada fungsi main() untuk menampilkan graf pada layar.

2. Guided 2

Source code

```
#include <iostream>

using namespace std;

/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;

    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

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

// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1;
    else
        return 0;
}
```

```

        // true

        // 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;

        }

    }

```

```

}

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

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

```

```

        inOrder(node->right);
    }
}

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

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {

```

```

        if (node != root)
        {
            node->parent->left = NULL;
            node->parent->right = NULL;
        }
        deleteTree(node->left);
        deleteTree(node->right);
        if (node == root)
        {
            delete root;
            root = NULL;
        }
        else
        {
            delete node;
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
    }
}

```



```

        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;

    }
}

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

// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
    }
}

```

```

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

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

```

```

        else
        {
            return heightKanan + 1;
        }
    }
}

// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() <<
endl;
}

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

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

    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);

```

```

nodeJ = insertRight('J', nodeG);
update('Z', nodeC);
update('C', nodeC);
retrieve(nodeC);
find(nodeC);
cout << "\n PreOrder :" << endl;
preOrder(root);
cout << "\n"
    << endl;
cout << " InOrder :" << endl;
inOrder(root);
cout << "\n"
    << endl;
cout << " PostOrder :" << endl;
postOrder(root);
cout << "\n"
    << endl;
charateristic();
deleteSub(nodeE);
cout << "\n PreOrder :" << endl;
preOrder();
cout << "\n"
    << endl;
charateristic();
}

```

Screenshot program

```
PS C:\Modul 5-9> g++ treeguided.cpp
PS C:\Modul 5-9> ./a

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
```

```
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:\Modul 5-9> █
```

Deskripsi Program

Program di atas merupakan implementasi dari Binary Tree Setiap simpul (node) memiliki maksimal dua anak, yaitu anak kiri (left child) dan anak kanan (right child).

1. Inisialisasi dan Cek Node: ``init()`` Menginisialisasi pohon (root) dengan nilai NULL dan ``isEmpty()`` untuk Memeriksa apakah pohon kosong atau tidak.
2. Pembuatan Node Baru: ``buatNode()`` untuk Membuat sebuah node baru sebagai root jika pohon masih kosong.
3. Penambahan Node Kiri dan Kanan: ``insertLeft()`` untuk Menambahkan node anak kiri pada sebuah node tertentu dan ``insertRight()`` untuk Menambahkan node anak kanan pada sebuah node tertentu.
4. Pengubahan Data pada Node: ``update()`` untuk Mengubah data pada sebuah node.
5. Pencarian dan Penampilan Data Node: ``retrieve()`` untuk Menampilkan data pada sebuah node dan ``find()`` untuk Menampilkan informasi lengkap tentang sebuah node, termasuk data, parent, sibling, dan child-nya.
6. Penelusuran (Traversal) Pohon: ``preOrder()`` untuk Melakukan penelusuran preOrder pada pohon (urutan: root-left-right) dan ``inOrder()`` untuk Melakukan penelusuran inOrder pada pohon (urutan: left-root-right) dan ``postOrder()`` untuk Melakukan penelusuran postOrder pada pohon (urutan: left-right-root).
7. Penghapusan Pohon dan Subtree: ``deleteTree()`` untuk Menghapus seluruh pohon dan ``deleteSub()`` untuk Menghapus sebuah subtree.
8. Informasi Karakteristik Pohon: ``size()`` untuk Menghitung jumlah node pada pohon, ``height()`` untuk Menghitung tinggi pohon (level tertinggi), dan ``characteristic()``: Menampilkan informasi tentang ukuran pohon (jumlah node), tinggi pohon, dan rata-rata node per level.

UNGUIDED

1. Unguided 1

```
//Shiva Indah Kurnia, 2311102035
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>

using namespace std;

int main()
{
    int Shiva_2311102035;
    cout << "Silahkan masukkan jumlah simpul: ";
    cin >> Shiva_2311102035;

    vector<string> simpul(Shiva_2311102035);
    vector<vector<int>> busur(Shiva_2311102035,
vector<int>(Shiva_2311102035, 0));

    cout << "Silahkan masukkan nama simpul " << endl;
    for (int i = 0; i < Shiva_2311102035; i++)
    {
        cout << "Simpul ke-" << (i + 1) << ": ";
        cin >> simpul[i];
    }

    cout << "Silahkan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < Shiva_2311102035; i++)
    {
        for (int j = 0; j < Shiva_2311102035; j++)
        {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> busur[i][j];
        }
    }

    cout << endl;
    cout << setw(7) << " ";
    for (int i = 0; i < Shiva_2311102035; i++)
    {
        cout << setw(8) << simpul[i];
    }
    cout << endl;
```

```

for (int i = 0; i < Shiva_2311102035; i++)
{
    cout << setw(7) << simpul[i];
    for (int j = 0; j < Shiva_2311102035; j++)
    {
        cout << setw(8) << busur[i][j];
    }
    cout << endl;
}
}

```

Screenshot program

```

PS C:\Modul 5-9> g++ graphunguided.cpp
PS C:\Modul 5-9> ./a
Silahkan masukkan jumlah simpul: 2
Silahkan masukkan nama simpul
Simpul ke-1: BALI
Simpul ke-2: PALU
Silahkan masukkan bobot antar simpul
BALI --> BALI = 0
BALI --> PALU = 3
PALU --> BALI = 4
PALU --> PALU = 0

      BALI  PALU
BALI    0    3
PALU    4    0
PS C:\Modul 5-9>

```

Deskripsi program

Program diatas adalah program yang dibuat untuk representasi graf menggunakan matriks. Program ini memungkinkan pengguna untuk memasukkan jumlah simpul, nama-nama simpul, dan bobot antar simpul dalam graf.

Berikut adalah penjelasan mengenai bagian-bagian dari program tersebut:

1. Baris 7-11: Mendefinisikan beberapa header file yang akan digunakan dalam program.

2. Baris 14: Mendeklarasikan variabel ``Fadhila_2211102195`` dengan tipe data ``int``. Nama variabel ini sepertinya tidak bermakna dan perlu diberi nama yang lebih deskriptif.

3. Baris 15-16: Meminta pengguna untuk memasukkan jumlah simpul yang akan dimasukkan ke dalam variabel ``Fadhila_2211102195``.

4. Baris 18-19: Membuat vector ``simpul`` dengan ukuran sebanyak ``Fadhila_2211102195`` untuk menyimpan nama-nama simpul.

5. Baris 20: Membuat matriks ``busur`` dengan ukuran ``Fadhila_2211102195`x`Fadhila_2211102195`` dan diinisialisasi dengan nilai 0 untuk menyimpan bobot antar simpul.

6. Baris 23-28: Meminta pengguna untuk memasukkan nama-nama simpul.

7. Baris 31-38: Meminta pengguna untuk memasukkan bobot antar simpul dengan mengiterasi setiap elemen matriks ``busur``.

8. Baris 41-46: Menampilkan matriks ketetanggaan yang sudah diisi dengan bobot antar simpul.

9. Baris 49-52: Menampilkan header kolom matriks ketetanggaan berupa nama simpul.

10. Baris 55-61: Menampilkan isi matriks ketetanggaan dengan nama simpul pada baris pertama dan bobot antar simpul pada setiap elemen matriks.

Program ini dapat digunakan untuk memodelkan graf dengan jumlah simpul yang ditentukan oleh pengguna dan mengisi bobot antar simpul dalam graf tersebut.

2. Unguided 2

```
//Shiva Indah Kurnia, 2311102035
#include <iostream>
using namespace std;

class BinarySearchTree
{
private:
    struct nodeTree
    {
        nodeTree *left;
        nodeTree *right;
        int data;
    };
    nodeTree *root;

public:
    BinarySearchTree()
    {
        root = NULL;
    }

    bool isEmpty() const { return root == NULL; }
    void print_inorder();
    void inorder(nodeTree *);
    void print_preorder();
    void preorder(nodeTree *);
    void print_postorder();
    void postorder(nodeTree *);
    void insert(int);
    void remove(int);
};

void BinarySearchTree::insert(int a)
{
    nodeTree *t = new nodeTree;
    nodeTree *parent;
    t->data = a;
    t->left = NULL;
    t->right = NULL;
    parent = NULL;

    if (isEmpty())
        root = t;
}
```

```

else
{
    nodeTree *current;
    current = root;

    while (current)
    {
        parent = current;
        if (t->data > current->data)
            current = current->right;
        else
            current = current->left;
    }

    if (t->data < parent->data)
        parent->left = t;
    else
        parent->right = t;
    }
}

void BinarySearchTree::remove(int a)
{
    // Mencari elemen yang akan dihapus
    bool found = false;
    if (isEmpty())
    {
        cout << "Pohon ini kosong!" << endl;
        return;
    }

    nodeTree *current;
    nodeTree *parent;
    current = root;

    while (current != NULL)
    {
        if (current->data == a)
        {
            found = true;
            break;
        }
        else
        {
            parent = current;

```

```

        if (a > current->data)
            current = current->right;
        else
            current = current->left;
    }
}
if (!found)
{
    cout << "Data tidak ditemukan!" << endl;
    return;
}

// Node dengan satu anak
if ((current->left == NULL && current->right != NULL) ||
(current->left != NULL && current->right == NULL))
{
    if (current->left == NULL && current->right != NULL)
    {
        if (parent->left == current)
        {
            parent->left = current->right;
            delete current;
        }
        else
        {
            parent->right = current->right;
            delete current;
        }
    }
    else
    {
        if (parent->left == current)
        {
            parent->left = current->left;
            delete current;
        }
        else
        {
            parent->right = current->left;
            delete current;
        }
    }
    return;
}
}

```

```

// Node tanpa anak
if (current->left == NULL && current->right == NULL)
{
    if (parent->left == current)
        parent->left = NULL;
    else
        parent->right = NULL;
    delete current;
    return;
}

// Node dengan dua anak
// Ganti node dengan nilai terkecil pada subtree sebelah
kanan
if (current->left != NULL && current->right != NULL)
{
    nodeTree *temp;
    temp = current->right;
    if ((temp->left == NULL) && (temp->right == NULL))
    {
        current = temp;
        delete temp;
        current->right = NULL;
    }
    else
    {
        if ((current->right)->left != NULL)
        {
            nodeTree *lcurrent;
            nodeTree *lcurrp;
            lcurrp = current->right;
            lcurrent = (current->right)->left;
            while (lcurrent->left != NULL)
            {
                lcurrp = lcurrent;
                lcurrent = lcurrent->left;
            }
            current->data = lcurrent->data;
            delete lcurrent;
            lcurrp->left = NULL;
        }
        else
        {
            nodeTree *tmp2;

```

```

        tmp2 = current->right;
        current->data = tmp2->data;
        current->right = tmp2->right;
        delete tmp2;
    }
}
return;
}
}

void BinarySearchTree::print_inorder()
{
    inorder(root);
}

void BinarySearchTree::inorder(nodeTree *b)
{
    if (b != NULL)
    {
        if (b->left)
            inorder(b->left);
        cout << " " << b->data << " ";
        if (b->right)
            inorder(b->right);
    }
    else
        return;
}

void BinarySearchTree::print_preorder()
{
    preorder(root);
}

void BinarySearchTree::preorder(nodeTree *b)
{
    if (b != NULL)
    {
        cout << " " << b->data << " ";
        if (b->left)
            preorder(b->left);
        if (b->right)
            preorder(b->right);
    }
    else

```

```

        return;
    }

void BinarySearchTree::print_postorder()
{
    postorder(root);
}

void BinarySearchTree::postorder(nodeTree *b)
{
    if (b != NULL)
    {
        if (b->left)
            postorder(b->left);
        if (b->right)
            postorder(b->right);
        cout << " " << b->data << " ";
    }
    else
        return;
}

int main()
{
    BinarySearchTree b;
    int ch, tmp, tmp1;
    while (1)
    {
        cout << endl
              << endl;
        cout << "-----" << endl;
        cout << "MENU OPERASI TREE BINARY SEARCH" << endl;
        cout << "-----" << endl;
        cout << "1. Penambahan/Pembuatan Pohon" << endl;
        cout << "2. Traversal In-Order" << endl;
        cout << "3. Traversal Pre-Order" << endl;
        cout << "4. Traversal Post-Order" << endl;
        cout << "5. Penghapusan" << endl;
        cout << "6. Keluar" << endl;
        cout << "Masukkan pilihan Anda: ";
        cin >> ch;
        switch (ch)
        {
            case 1:
                cout << "Masukkan Angka yang akan ditambahkan: ";

```

```

        cin >> tmp;
        b.insert(tmp);
        break;
    case 2:
        cout << endl;
        cout << "Traversal In-Order" << endl;
        cout << "-----" << endl;
        b.print_inorder();
        break;
    case 3:
        cout << endl;
        cout << "Traversal Pre-Order" << endl;
        cout << "-----" << endl;
        b.print_preorder();
        break;
    case 4:
        cout << endl;
        cout << "Traversal Post-Order" << endl;
        cout << "-----" << endl;
        b.print_postorder();
        break;
    case 5:
        cout << "Masukkan angka yang akan dihapus: ";
        cin >> tmp1;
        b.remove(tmp1);
        break;
    case 6:
        return 0;
    }
}
}

```

Screenshot program

```

-----
MENU OPERASI TREE BINARY SEARCH
-----
1. Penambahan/Pembuatan Pohon
2. Traversal In-Order
3. Traversal Pre-Order
4. Traversal Post-Order
5. Penghapusan
6. Keluar
Masukkan pilihan Anda: 1
Masukkan Angka yang akan ditambahkan: 1

```



```
07. Keluar
Masukkan pilihan Anda: 1
Masukkan Angka yang akan ditambahkan: 2
```

```
Masukkan pilihan Anda: 1
Masukkan Angka yang akan ditambahkan: 3
```

```
Masukkan pilihan Anda: 1
Masukkan Angka yang akan ditambahkan: 4
```

```
Masukkan pilihan Anda: 1
Masukkan Angka yang akan ditambahkan: 7
```

```
Masukkan pilihan Anda: 1
Masukkan Angka yang akan ditambahkan: 8
```

```
Masukkan pilihan Anda: 2

Traversal In-Order
-----
1 2 3 4 7 8
```

```
08. Keluar
Masukkan pilihan Anda: 3

Traversal Pre-Order
-----
1 2 3 4 7 8
```

```
Masukkan pilihan Anda: 4

Traversal Post-Order
-----
8 7 4 3 2 1
```

```
Masukkan pilihan Anda: 5
Masukkan angka yang akan dihapus: 8
```

```
Masukkan pilihan Anda: 2

Traversal In-Order
-----
1 2 3 4 7
```

Deskripsi program

Program di atas adalah implementasi dari Binary Search Tree (BST) menggunakan C++. Program diatas terdiri dari:

1. Deklarasi Kelas `BinarySearchTree` Kelas ini berfungsi untuk merepresentasikan Binary Search Tree, Memiliki tiga atribut pribadi: `nodeTree *left` dan `nodeTree *right`: Pointer ke simpul anak kiri dan anak kanan, `int data`: Nilai data yang disimpan di dalam simpul, Memiliki satu atribut publik: `nodeTree *root`: Pointer ke simpul akar pohon BST.
2. Konstruktor `BinarySearchTree`: Inisialisasi pointer `root` dengan nilai NULL untuk menandakan bahwa pohon masih kosong.
3. Fungsi `isEmpty`: Memeriksa apakah pohon kosong atau tidak.
4. Fungsi `insert`: Menyisipkan elemen baru ke dalam BST dengan mempertimbangkan aturan BST, Jika pohon kosong, elemen baru menjadi akar. Jika tidak, elemen baru akan ditempatkan sesuai dengan perbandingan nilainya dengan elemen-elemen yang sudah ada.
5. Fungsi `remove`: Menghapus elemen tertentu dari BST, Pertama, mencari elemen yang akan dihapus dengan melakukan traversal pohon, Jika elemen ditemukan, pilihan penghapusan dilakukan berdasarkan jumlah anak simpul tersebut (tidak ada anak, satu anak, atau dua anak), Jika simpul memiliki satu anak, simpul tersebut digantikan oleh anaknya, Jika simpul memiliki dua anak, simpul digantikan oleh nilai terkecil pada subtree kanan.
6. Fungsi `print_inorder`, `print_preorder`, dan `print_postorder`: Mencetak elemen-elemen pohon dalam urutan In-Order, Pre-Order, dan Post-Order secara rekursif, Memanggil fungsi rekursif yang sesuai (`inorder`, `preorder`, dan `postorder`) dengan memberikan simpul saat ini sebagai argumen.
7. Fungsi `inorder`, `preorder`, dan `postorder`: Fungsi rekursif yang digunakan untuk traversal dan pencetakan elemen-elemen pohon dalam urutan In-Order, Pre-Order, dan Post-Order, Mengecek terlebih dahulu apakah simpul tidak NULL, Jika tidak, maka pemanggilan rekursif dilakukan pada simpul anak kiri, simpul sendiri, dan simpul anak kanan sesuai urutan traversal yang diinginkan.
8. Fungsi `main`: Membuat objek `BinarySearchTree` baru, Menampilkan menu pilihan operasi pada BST dan menerima input pengguna, Memanggil fungsi-fungsi yang sesuai berdasarkan pilihan pengguna.

KESIMPULAN

Graph dan pohon adalah struktur data penting dalam pemrograman. Graph digunakan untuk menunjukkan hubungan antara objek atau entitas. Pohon adalah jenis diagram khusus dengan hierarki atau struktur hierarkis. Konsep pointer dan struktur data rekursif digunakan saat mengimplementasikan bagan dan pohon di C++. Pointer digunakan untuk menghubungkan titik-titik dalam diagram atau pohon. Struktur data rekursif memungkinkan operasi rekursif seperti traversal dan pencarian elemen. Binary Search Tree (BST) adalah implementasi khusus dari pohon yang efisien untuk mencari data. Dalam BST ada aturan bahwa elemen di sebelah kiri harus lebih kecil dari elemen di sebelah kanan. Aturan ini memungkinkan pencarian data dengan kompleksitas $O(\log n)$, dimana n adalah jumlah elemen dalam BST. Implementasi grafik dan pohon dalam C++ dapat dilakukan dengan menggunakan struktur data dan operasi rekursif. Pada Pohon kita juga mempelajari bagaimana pengimplementasian transerval inorder, preorder dan post order pada pemrograman C++