LAPORAN PRAKTIKUM

MODUL IX
GRAPH DAN TREE

Disusun oleh:
Syafanida Khakiki
NIM: 2311102005

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

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

BAB I

TUJUAN PRAKTIKUM

a. Mahasiswa diharapkan mampu memahami graph dan tree
b. Mahasiswa diharapkan mampu mengimplementasikan graph dan tree pada pemrograman

BAB II
DASAR TEORI

1. Graph

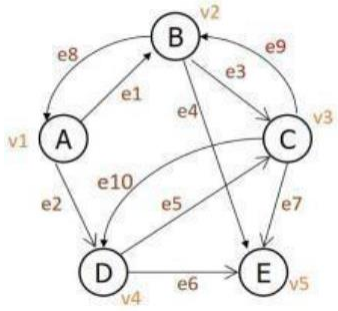
Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk edge atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G=(V,E)$$

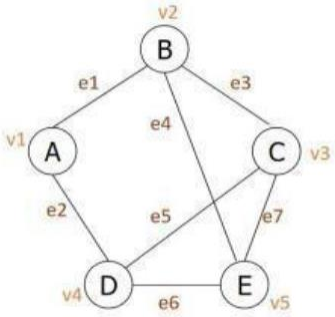
Dimana G adalah Graph, V adalah simpul atau vertex dan node sebagai titik atau egde.

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

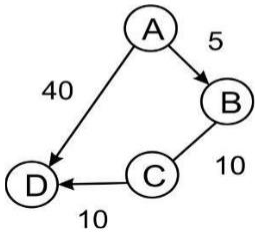
Jenis-jenis Graph



Directed Graph



Undirected Graph



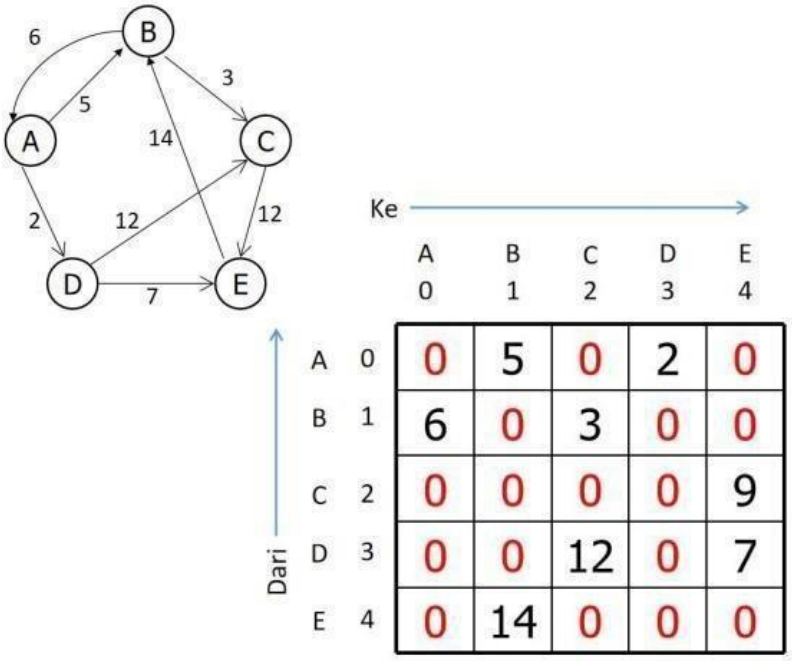
Weight Graph

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.

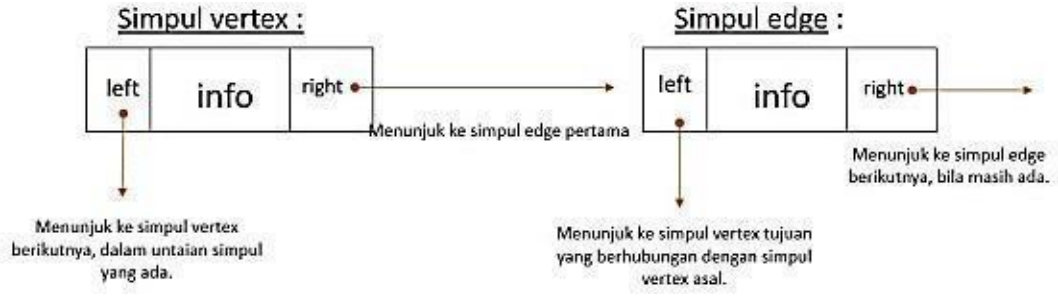
c. **Weigth Graph :** Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph Representasi dengan Matriks



Gambar 4 Representasi Graph dengan Matriks

Representasi dengan Linked List

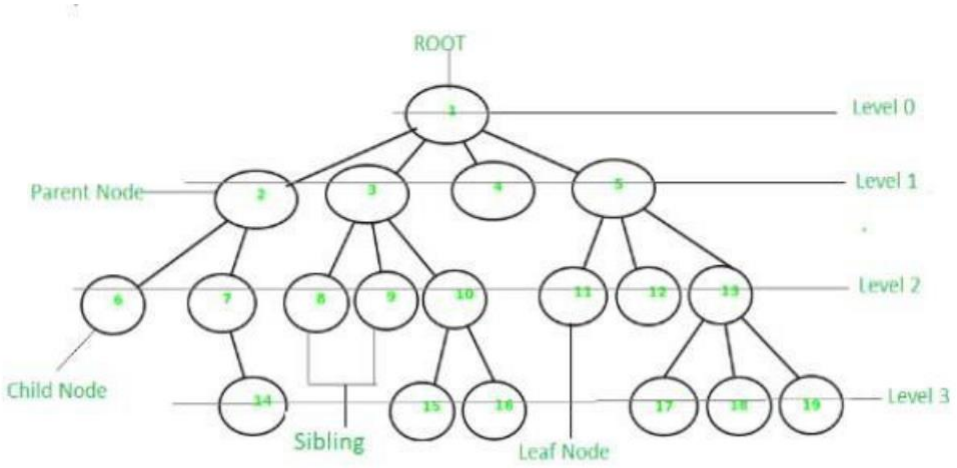


Gambar 5 Representasi Graph dengan Linked List

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.

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 :

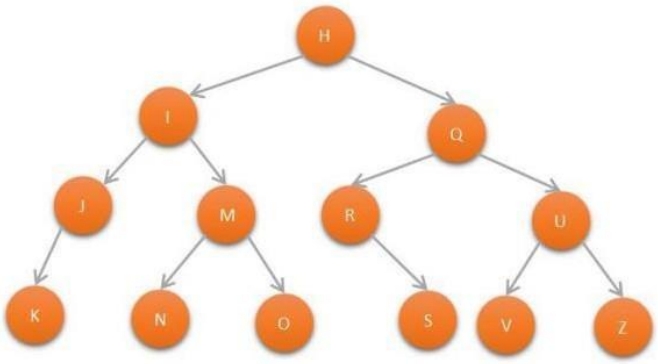


Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor 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 diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.

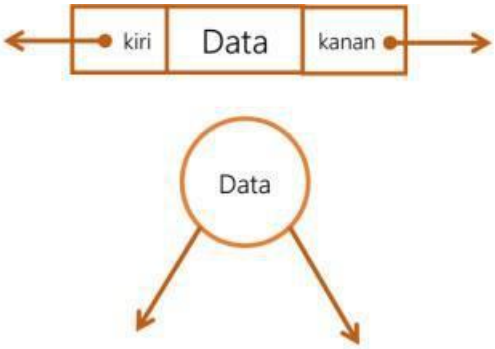
Gambar 1, menunjukkan contoh dari struktur data binary tree.



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

- **Create**: digunakan untuk membentuk binary tree baru yang masih kosong.
- **Clear**: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
- **isEmpty**: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
- **Insert**: digunakan untuk memasukkan sebuah node kedalam tree.

- **Find**: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong**.**
- **Update**: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

- **Retrive**: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Delete Sub**: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
- **Characteristic**: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
- **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.

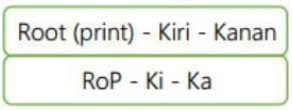
1. Pre-Order

Penelusuran secara pre-order memiliki alur:

Cetak data pada simpul root

Secara rekursif mencetak seluruh data pada subpohon kiri

Secara rekursif mencetak seluruh data pada subpohon kanan Dapat kita turunkan rumus penelusuran menjadi:



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:

3. Post Order

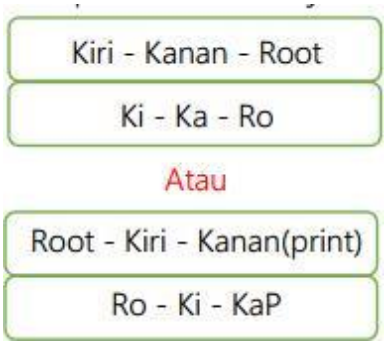
Penelusuran secara in-order memiliki alur:

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

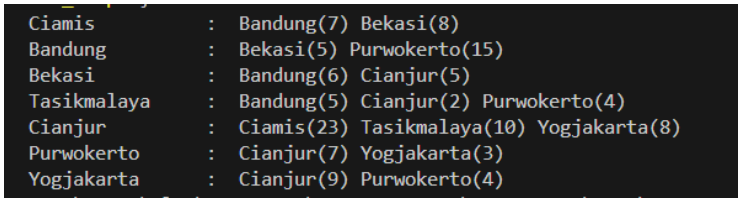
GUIDED

1. **Guided 1**

   Program Graph :

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

Output :



Deskripsi program :

Program ini menampilkan representasi grafik terarah yang menghubungkan 7 kota di Jawa Barat.

Langkah-langkah program:

1. **Mendefinisikan data:**
   - Program mendefinisikan array `simpul` yang berisi nama 7 kota di Jawa Barat:
     - `Ciamis`
     - `Bandung`
     - `Bekasi`
     - `Tasikmalaya`
     - `Cianjur`
     - `Purwokerto`
     - `Yogyakarta`
   - Program mendefinisikan array 2 dimensi `busur` yang merepresentasikan bobot koneksi antar kota. Bobot menunjukkan jarak antar kota:
   - `{`
   - `{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}`
   - `}`
2. **Membuat fungsi `tampilGraph`:**
   - Fungsi `tampilGraph` mendefinisikan format tabel untuk menampilkan representasi grafik.
   - Fungsi ini mengiterasi baris dan kolom array `busur`.
   - Jika bobot (nilai pada `busur[baris][kolom]`) tidak sama dengan 0, maka:
     - Nama kota tujuan (dari `simpul[kolom]`) dan bobotnya (dari `busur[baris][kolom]`) ditampilkan dalam format tabel.
   - Fungsi ini kemudian menampilkan baris tabel berikutnya.
3. **Memanggil fungsi `tampilGraph`:**
   - Fungsi `tampilGraph` dipanggil untuk menampilkan representasi grafik yang telah didefinisikan.

Penjelasan representasi grafik:

- Grafik direpresentasikan sebagai array 2 dimensi `busur`.
- Baris pada array mewakili kota asal, dan kolom mewakili kota tujuan.

- Nilai pada `busur[baris][kolom]` menunjukkan bobot koneksi (jarak) antara kota asal (baris) dan kota tujuan (kolom).
- Nilai 0 menunjukkan tidak adanya koneksi antar kota.
- Fungsi `tampilGraph` memformat representasi grafik menjadi tabel yang lebih mudah dibaca, di mana:
    - Kolom pertama menunjukkan nama kota asal.
    - Kolom berikutnya menunjukkan kota tujuan yang terhubung dengan kota asal, beserta jaraknya dalam tanda kurung.

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

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

```
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil dihapus." <<
endl;
    }
}
// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}
// Cek Height Level Tree
int height(Pohon *node = root)
```

```
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}
// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}
int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
```

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

Output :

```
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
```

Deskripsi program :

Program ini mengimplementasikan struktur data pohon binary dan berbagai operasi yang dapat dilakukan pada pohon tersebut.

Struktur Data Pohon Binary:

- Pohon binary adalah struktur data hierarki, di mana setiap node (simpul) memiliki maksimal 2 anak (child), yaitu anak kiri dan anak kanan.
- Program ini menggunakan struktur `Pohon` untuk merepresentasikan node pada pohon binary. Struktur ini memiliki:
  - `data`: karakter yang disimpan pada node.
  - `left`: pointer ke node anak kiri.
  - `right`: pointer ke node anak kanan.
  - `parent`: pointer ke node parent (induk).
- Variabel `root` merupakan pointer yang menunjuk ke node root (akar) pada pohon binary.

Operasi pada Pohon Binary:

Program ini menyediakan berbagai fungsi untuk melakukan operasi pada pohon binary:

- **Inisialisasi:**
  - `init()`: fungsi untuk menginisialisasi pohon binary menjadi kosong (root bernilai NULL).
- **Mengecek Kondisi Pohon:**
  - `isEmpty()`: fungsi untuk mengecek apakah pohon binary kosong (root bernilai NULL).
- **Membuat Node Baru:**
  - `buatNode(char data)`: fungsi untuk membuat node baru dengan data karakter tertentu dan menjadikannya sebagai root pohon jika pohon masih kosong.
- **Menambah Node Anak:**
  - `insertLeft(char data, Pohon *node)`: fungsi untuk menambah node anak kiri dengan data karakter tertentu pada node tertentu.
  - `insertRight(char data, Pohon *node)`: fungsi untuk menambah node anak kanan dengan data karakter tertentu pada node tertentu.
- **Mengubah Data Node:**
  - `update(char data, Pohon *node)`: fungsi untuk mengubah data karakter pada node tertentu.
- **Melihat Isi Data Node:**
  - `retrieve(Pohon *node)`: fungsi untuk melihat data karakter pada node tertentu.
- **Mencari Informasi Node:**

    - `find(Pohon *node)`: fungsi untuk menampilkan informasi detail dari node tertentu, seperti data, root, parent, sibling (saudara), dan child (anak).
- **Penelusuran (Traversal):**
    - Program ini menyediakan fungsi untuk melakukan penelusuran pohon dengan 3 metode:
        - `preOrder(Pohon *node)`: penelusuran pre-order, mengunjungi node root terlebih dahulu, kemudian anak kiri dan kanan secara rekursif.
        - `inOrder(Pohon *node)`: penelusuran in-order, mengunjungi anak kiri terlebih dahulu, kemudian node root, dan terakhir anak kanan secara rekursif.
        - `postOrder(Pohon *node)`: penelusuran post-order, mengunjungi anak kiri dan kanan terlebih dahulu secara rekursif, kemudian node root.
- **Penghapusan:**
    - `deleteTree(Pohon *node)`: fungsi untuk menghapus seluruh pohon binary secara rekursif.
    - `deleteSub(Pohon *node)`: fungsi untuk menghapus subtree (subpohon) yang berakar pada node tertentu.
    - `clear()`: fungsi untuk menghapus seluruh pohon binary dan menginisialisasi root menjadi NULL.
- **Informasi Karakteristik Pohon:**
    - `size(Pohon *node)`: fungsi untuk menghitung jumlah node pada pohon binary.
    - `height(Pohon *node)`: fungsi untuk menghitung tinggi (level) pohon binary.
    - `charateristic()`: fungsi untuk menampilkan karakteristik pohon binary, seperti jumlah node (size), tinggi (height), dan rata-rata node per level.

BAB IV
UNGUIDED

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

Source code :

```
#include <iostream>
#include <vector>
#include <limits.h>

using namespace std;

// Function to print the adjacency matrix
void printMatrix(vector<vector<int>> &matrix, vector<string> &cities) {
    cout << "\t";
    for (const auto &city : cities) {
        cout << city << "\t";
    }
    cout << endl;

    for (int i = 0; i < matrix.size(); ++i) {
        cout << cities[i] << "\t";
        for (int j = 0; j < matrix[i].size(); ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

int main() {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;

    vector<string> cities(n);
    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> cities[i];
    }

    vector<vector<int>> adjMatrix(n, vector<int>(n, 0));
```

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

    printMatrix(adjMatrix, cities);

    return 0;
}
```

Output :

```
Silakan masukan jumlah simpul: 4
Silakan masukan nama simpul
Simpul 1: Purwokerto
Simpul 2: Purbalingga
Simpul 3: Banjarnegara
Simpul 4: Wonosobo
Silakan masukan bobot antar simpul
Purwokerto ---> Purbalingga: 10
Purwokerto ---> Banjarnegara: 15
Purwokerto ---> Wonosobo: 20
Purbalingga ---> Purwokerto: 10
Purbalingga ---> Banjarnegara: 10
Purbalingga ---> Wonosobo: 15
Banjarnegara ---> Purwokerto: 15
Banjarnegara ---> Purbalingga: 10
Banjarnegara ---> Wonosobo: 10
Wonosobo ---> Purwokerto: 20
Wonosobo ---> Purbalingga: 15
Wonosobo ---> Banjarnegara: 10
          Purwokerto      Purbalingga     Banjarnegara    Wonosobo
Purwokerto      0       10      15      20
Purbalingga     10      0       10      15
Banjarnegara    15      10      0       10
Wonosobo        20      15      10      0
```

Deskripsi Program :

Program ini ditulis dalam bahasa C++ dan digunakan untuk merepresentasikan graf menggunakan matriks adjasensi.

Komponen Program:

2. **`printMatrix` Function:**
    - Mencetak matriks adjasensi ke konsol.
    - Baris pertama berisi nama-nama kota (simpul) dalam graf.
    - Kolom pertama berisi nama-nama kota (simpul) dalam graf.
    - Elemen lainnya dalam matriks menunjukkan jarak (bobot) antar kota (simpul).

3. **`main` Function:**
    - Meminta user untuk memasukkan jumlah kota (simpul) dalam graf (`n`).
    - Membaca nama-nama kota (simpul) dari user dan menyimpannya dalam vektor `cities`.
    - Memintialisiasi matriks adjasensi (`adjMatrix`).
        - Matriks ini bertipe `vector<vector<int>>`.
        - Setiap elemen dalam matriks bertipe `int` yang merepresentasikan jarak (bobot) antar kota.
    - Meminta user untuk memasukkan bobot (jarak) antar kota (simpul).
        - Looping melalui semua pasangan kota yang mungkin (kecuali diagonal utama yang selalu bernilai 0).
        - Membaca bobot (jarak) yang dimasukkan user dan menyimpannya ke dalam elemen yang sesuai pada matriks adjasensi.
    - Memanggil fungsi `printMatrix` untuk menampilkan matriks adjasensi ke konsol.

2. Unguided 2

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

Source Code :

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

void printMatrix(vector<vector<int>> &matrix, vector<string> &cities) {
    cout << "\t";
    for (const auto &city : cities) {
        cout << city << "\t";
    }
    cout << endl;

    for (int i = 0; i < matrix.size(); ++i) {
        cout << cities[i] << "\t";
        for (int j = 0; j < matrix[i].size(); ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

void inputTreeData(vector<string> &cities, vector<vector<int>> &adjMatrix) {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
```

```
    cin >> n;

    cities.resize(n);
    adjMatrix.resize(n, vector<int>(n, 0));

    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> cities[i];
    }

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

int main() {
    vector<string> cities;
    vector<vector<int>> adjMatrix;
    int choice;

    do {
        cout << "\nMenu:\n";
        cout << "1. Input data tree\n";
        cout << "2. Tampilkan matrix adjacency\n";
        cout << "3. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                inputTreeData(cities, adjMatrix);
                break;
            case 2:
                if (cities.empty() || adjMatrix.empty()) {
                    cout << "Data tree belum diinputkan.\n";
                } else {
                    printMatrix(adjMatrix, cities);
                }
```

```
                break;
            case 3:
                cout << "Keluar dari program.\n";
                break;
            default:
                cout << "Pilihan tidak valid. Silakan coba lagi.\n";
        }
    } while (choice != 3);

    return 0;
}
```

Output :

```
Menu:
1. Input data tree
2. Tampilkan matrix adjacency
3. Keluar
Pilihan Anda: 1
Silakan masukan jumlah simpul: 3
Silakan masukan nama simpul
Simpul 1: jakarta
Simpul 2: bandung
Simpul 3: purwokerto
Silakan masukan bobot antar simpul
jakarta ---> bandung: 20
jakarta ---> purwokerto: 30
bandung ---> jakarta: 20
bandung ---> purwokerto: 30
purwokerto ---> jakarta: 30
purwokerto ---> bandung: 20

Menu:
1. Input data tree
2. Tampilkan matrix adjacency
3. Keluar
Pilihan Anda: 2
        jakarta bandung purwokerto
jakarta 0       20      30
bandung 20      0       30
purwokerto      30      20      0
```

Deskripsi Program :

Program ini ditulis dalam bahasa C++ dan menyediakan antarmuka menu interaktif untuk merepresentasikan graf menggunakan matriks adjasensi.

Komponen Program:

- **`printMatrix` Function:**
  - Sama seperti program sebelumnya, fungsi ini mencetak matriks adjasensi ke konsol.
- **`inputTreeData` Function:**
  - Meminta user untuk memasukkan jumlah kota (simpul) dalam graf (`n`).

    - Membaca nama-nama kota (simpul) dari user dan menyimpannya dalam vektor `cities`.
    - Memintialisiasi matriks adjasensi (`adjMatrix`).
    - Meminta user untuk memasukkan bobot (jarak) antar kota (simpul).
        - Looping melalui semua pasangan kota yang mungkin (kecuali diagonal utama yang selalu bernilai 0).
        - Membaca bobot (jarak) yang dimasukkan user dan menyimpannya ke dalam elemen yang sesuai pada matriks adjasensi.
- **`main` Function:**
    - Deklarasi variabel untuk menyimpan nama kota (simpul) dan matriks adjasensi.
    - Menampilkan menu interaktif yang memungkinkan user untuk:
        - **1. Input Data Tree (Pohon)**: Memanggil fungsi `inputTreeData` untuk memasukkan informasi graf.
            - **Catatan:** Program ini menggunakan istilah "tree" (pohon) tetapi fungsinya tetap representasi graf secara umum.
        - **2. Tampilkan Matrix Adjacency (Tampilkan Matriks Adjasensi)**: Memanggil fungsi `printMatrix` untuk menampilkan matriks adjasensi ke konsol, dengan pengecekan terlebih dahulu apakah data graf sudah diinputkan.
        - **3. Keluar (Keluar dari program)**: Menutup program.

BAB IV

KESIMPULAN

Graph

1. **Definisi dan Struktur:**
   - **Graph** adalah struktur data yang terdiri dari simpul (nodes atau vertices) dan sisi (edges) yang menghubungkan pasangan simpul.
   - Graph dapat bersifat berarah (directed graph) atau tidak berarah (undirected graph).
   - Graph sering diimplementasikan menggunakan matriks ketetanggaan (adjacency matrix) atau daftar ketetanggaan (adjacency list).
2. **Penggunaan:**
   - Graph digunakan untuk merepresentasikan hubungan antara objek, seperti jaringan sosial, jaringan komputer, peta, dan berbagai aplikasi lainnya.
   - Dalam pemrograman, graph dapat digunakan untuk menemukan jalur terpendek, melakukan pencarian lebar dulu (BFS) dan pencarian dalam dulu (DFS), serta menyelesaikan masalah pengoptimalan.
3. **Implementasi dalam Program:**
   - Input data graph dapat melibatkan pengisian jumlah simpul, nama simpul, dan bobot antar simpul.
   - Adjacency matrix digunakan untuk merepresentasikan hubungan antar simpul dalam program, dimana `matrix[i][j]` menunjukkan bobot dari simpul `i` ke simpul `j`.
   - Program dapat memberikan antarmuka pengguna untuk memasukkan data dan melihat representasi graph dalam bentuk matriks.

Tree

1. **Definisi dan Struktur:**
   - **Tree** adalah jenis khusus dari graph yang tidak mengandung siklus dan memiliki satu simpul akar (root node) dari mana semua simpul lainnya dapat dicapai.
   - Setiap simpul dalam tree memiliki nol atau lebih anak (children), dan setiap anak hanya memiliki satu induk (parent).
2. **Penggunaan:**
   - Tree digunakan dalam berbagai aplikasi seperti struktur data (binary search tree, heap), representasi hierarki (struktur organisasi, sistem file), dan banyak algoritma pengoptimalan.
   - Tree traversal (penelusuran tree) seperti preorder, inorder, dan postorder adalah operasi umum dalam pengolahan tree.
3. **Implementasi dalam Program:**
   - Input data tree biasanya melibatkan pengisian simpul dan hubungan induk-anak antar simpul.
   - Tree dapat direpresentasikan menggunakan adjacency list atau struktur khusus seperti linked nodes.
   - Program yang melibatkan tree dapat menyediakan fungsionalitas untuk memasukkan data tree, menampilkan struktur tree, dan melakukan operasi penelusuran.

Perbandingan dan Hubungan antara Graph dan Tree

1. **Graph:**
    - Umum, bisa berarah atau tidak berarah.
    - Dapat memiliki siklus.
    - Mempunyai representasi yang lebih kompleks dengan berbagai tipe hubungan antar simpul.
2. **Tree:**
    - Jenis khusus dari graph tanpa siklus.
    - Memiliki struktur hierarkis dengan satu akar.
    - Digunakan untuk representasi yang lebih terstruktur dan teratur.
3. **Implementasi dan Penggunaan:**
    - Graph dan tree keduanya penting dalam pemrograman dan ilmu komputer.
    - Keduanya dapat diimplementasikan menggunakan struktur data seperti adjacency matrix dan adjacency list.
    - Pilihan antara menggunakan graph atau tree bergantung pada kebutuhan spesifik aplikasi dan sifat data yang direpresentasikan.

DAFTAR PUSTAKA

Asisten Praktikum (2024). Search. Learning Management System.

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications.