LAPORAN PRAKTIKUM
Modul XIV
“GRAF”

Disusun Oleh:
Zivana Afra Yulianto -2211104039
SE-07-02

Dosen:
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

Tujuan praktikum ini adalah:

1. Mempelajari konsep graf menggunakan adjacency matrix.
2. Mengimplementasikan graf berbobot dan tidak berarah menggunakan adjacency matrix.
3. Memahami penggunaan struktur data untuk hubungan antar simpul dalam graf.
4. Membuat program untuk menampilkan adjacency matrix dengan input interaktif.

2. Landasan Teori

Graf
Graf terdiri dari simpul (nodes) dan sisi (edges) yang menghubungkan simpul-simpul tersebut. Jenis-jenis graf:

- Berarah (Directed): Sisi memiliki arah.
- Tidak Berarah (Undirected): Sisi tanpa arah.
- Berbobot (Weighted): Sisi memiliki bobot.

Adjacency Matrix
Representasi graf dalam bentuk matriks 2D, di mana elemen matriks pada baris `i` dan kolom `j` menunjukkan hubungan antar simpul. Untuk graf berbobot, nilai elemen menunjukkan bobot sisi. Untuk graf tidak berarah, matriks simetris.

Traversal Graf

- DFS (Depth First Search): Menjelajahi graf secara mendalam.
- BFS (Breadth First Search): Menjelajahi graf secara lebar.

C++ dan Graf
C++ mendukung penggunaan struktur data seperti array dan vector untuk membangun graf dan melakukan traversal.

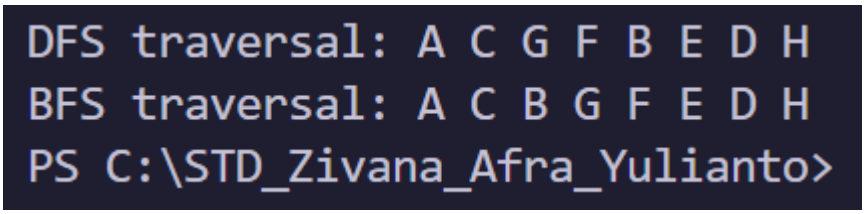
3. Guided

GUIDED I :

```
#include <iostream>
#include <queue>

using namespace std;

struct ElmNode;

struct ElmEdge {
    ElmNode *Node;
    ElmEdge *Next;
};

struct ElmNode {
    char info;
    bool visited;
    ElmEdge *firstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *first;
};

void CreateGraph(Graph &G) {
    G.first = NULL;
}

void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode;
    newNode->info = X;
    newNode->visited = false;
    newNode->firstEdge = NULL;
    newNode->Next = NULL;

    if (G.first == NULL) {
        G.first = newNode;
    } else {
        ElmNode *temp = G.first;
        while (temp->Next != NULL) {
            temp = temp->Next;
        }
        temp->Next = newNode;
    }
}

void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge;
    newEdge->Node = N2;
    newEdge->Next = N1->firstEdge;
    N1->firstEdge = newEdge;
}

void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " ";
        temp = temp->Next;
    }
    cout << endl;
}

void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false;
        temp = temp->Next;
    }
}
```

```
void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return;
    }
    N->visited = true;
    cout << N->info << " ";
    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node);
        }
        edge = edge->Next;
    }
}

void PrintBFS(Graph G, ElmNode *N) {
    queue<ElmNode*> q;
    q.push(N);
    N->visited = true;

    while (!q.empty()) {
        ElmNode *current = q.front();
        q.pop();
        cout << current->info << " ";

        ElmEdge *edge = current->firstEdge;
        while (edge != NULL) {
            if (!edge->Node->visited) {
                edge->Node->visited = true;
                q.push(edge->Node);
            }
            edge = edge->Next;
        }
    }
}

int main() {
    Graph G;
    CreateGraph(G);

    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    ElmNode *A = G.first;
    ElmNode *B = A->Next;
    ElmNode *C = B->Next;
    ElmNode *D = C->Next;
    ElmNode *E = D->Next;
    ElmNode *F = E->Next;
    ElmNode *G1 = F->Next;
    ElmNode *H = G1->Next;

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(D, H);

    cout << "DFS traversal: ";
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;

    cout << "BFS traversal: ";
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}
```

SCREENSHOOT OUTPUT :



DESKRIPSI :

Program ini merepresentasikan graf menggunakan adjacency list dengan struktur data dinamis di C++. Program dapat melakukan traversal graf menggunakan DFS (Depth-First Search) dan BFS (Breadth-First Search).

Fungsi Utama:

1. `CreateGraph`: Membuat graf kosong.
2. `InsertNode`: Menambahkan simpul ke dalam graf.
3. `ConnectNode`: Menyambungkan dua simpul dengan sisi.
4. `PrintDFS`: Traversal graf secara mendalam menggunakan rekursi.
5. `PrintBFS`: Traversal graf secara melintang menggunakan queue.
6. `ResetVisited`: Mengatur ulang status "visited" untuk traversal baru.

4. Unguided

UNGUIDED 1 :

```
#include <iostream>
#include <vector>
#include <iomanip> // Untuk format output
using namespace std;

int main()
{
    int numNodes;

    // Input jumlah simpul
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> numNodes;

    // Input nama simpul
    vector<string> nodes(numNodes);
    for (int i = 0; i < numNodes; i++)
    {
        cout << "Simpul " << i + 1 << ": ";
        cin >> nodes[i];
    }

    // Membuat matriks bobot
    vector<vector<int>> graph(numNodes, vector<int>(numNodes, 0));

    // Input bobot antar simpul
    cout << "Silakan masukkan bobot antar simpul:\n";
    for (int i = 0; i < numNodes; i++)
    {
```

```
        {
            if (i == j)
            {
                graph[i][j] = 0; // Bobot ke diri sendiri adalah 0
            }
            else
            {
                cout << nodes[i] << " --> " << nodes[j] << " = ";
                cin >> graph[i][j];
            }
        }
    }

    // Menampilkan matriks adjacency
    cout << "\nAdjacency Matrix:\n";
    cout << setw(10) << ""; // Header kosong
    for (const auto &node : nodes)
    {
        cout << setw(10) << node;
    }
    cout << endl;

    for (int i = 0; i < numNodes; i++)
    {
        cout << setw(10) << nodes[i];
        for (int j = 0; j < numNodes; j++)
        {
            cout << setw(10) << graph[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

SCREENSHOOT OUTPUT :

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

Adjacency Matrix:
                 BALI      PALU
      BALI          0         3
      PALU          4         0

Process returned 0 (0x0)   execution time : 14.110 s
Press any key to continue.
```

][

DESKRIPSI :

Program ini merepresentasikan graf berbobot menggunakan adjacency matrix di C++. Pengguna dapat memasukkan jumlah simpul, nama simpul, dan bobot antar simpul. Program kemudian menampilkan matriks adjacency berdasarkan input tersebut.
Fungsi Utama:

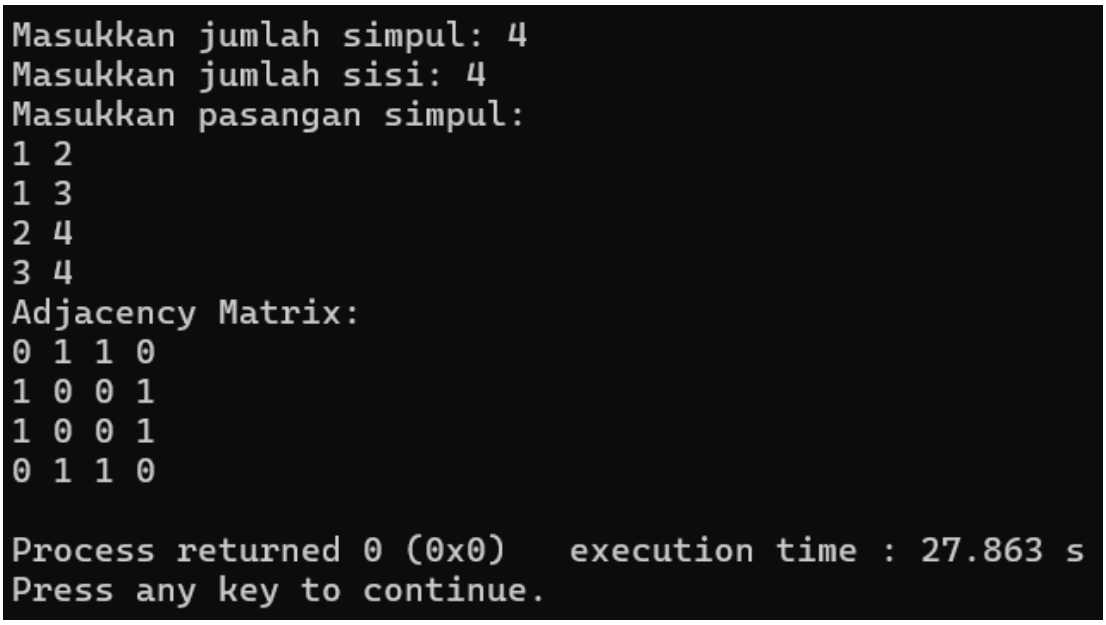
1. Input Simpul:
    - Meminta jumlah simpul dan nama-nama simpul dari pengguna.
2. Input Bobot Antar Simpul:
    - Meminta bobot antara setiap pasangan simpul.
    - Bobot dari simpul ke dirinya sendiri otomatis diatur ke `0`.
3. Tampilkan Matriks Adjacency:
    - Menampilkan matriks adjacency dalam format tabel, dengan header berupa nama simpul.

UNDUIDED 2 :

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

void printMatrix(const vector<vector<int>> &matrix)
{
    for (const auto &row : matrix)
    {
        for (const auto &val : row)
        {
            cout << val << " ";
        }
        cout << endl;
    }
}

int main()
{
    int numVertices, numEdges;

    // Input jumlah simpul dan sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> numVertices;
    cout << "Masukkan jumlah sisi: ";
    cin >> numEdges;

    // Inisialisasi adjacency matrix
    vector<vector<int>> adjMatrix(numVertices, vector<int>(numVertices, 0));

    // Input pasangan simpul
    cout << "Masukkan pasangan simpul:" << endl;
    for (int i = 0; i < numEdges; i++)
    {
        int u, v;
        cin >> u >> v;

        // Simpul diubah menjadi indeks (0-based)
        u--;
        v--;

        // Tandai sisi dalam adjacency matrix
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1; // Karena graf tidak berarah
    }
```

```
    // Output adjacency matrix
    cout << "Adjacency Matrix:" << endl;
    printMatrix(adjMatrix);

    return 0;
}
```

Screenshoot output :



Deskripsi :

Program ini merepresentasikan graf tidak berarah menggunakan adjacency matrix di C++. Pengguna dapat memasukkan jumlah simpul, jumlah sisi, dan pasangan simpul yang terhubung oleh sisi. Matriks adjacency kemudian ditampilkan sebagai hasil.

Fungsi Utama:

1. Input Jumlah Simpul dan Sisi:
    - Meminta jumlah simpul (vertices) dan jumlah sisi (edges) dari pengguna.
2. Input Pasangan Simpul:
    - Pengguna memasukkan pasangan simpul yang memiliki sisi (terhubung).
    - Program menggunakan indeks 0-based (dikurangi 1 dari input).
3. Adjacency Matrix:
    - Matriks diinisialisasi dengan nilai `0` (tidak ada hubungan).
    - Jika ada sisi antara simpul `u` dan `v`, elemen `adjMatrix[u][v]` dan `adjMatrix[v][u]` diatur menjadi `1` (graf tidak berarah).
4. Output Matriks:
    - Menampilkan adjacency matrix, yang menunjukkan hubungan antar simpul.

5. Kesimpulan

Kesimpulan praktikum ini:

1. Adjacency Matrix efektif untuk graf dengan jumlah simpul yang tidak terlalu besar.
2. Program mampu menampilkan adjacency matrix dan melakukan traversal graf (DFS, BFS).
3. Konsep graf dapat diterapkan dalam berbagai kasus nyata seperti pemetaan rute dan analisis jaringan.
4. C++ memberikan fleksibilitas dalam membangun struktur graf.
5. Praktikum ini memperdalam pemahaman tentang representasi dan traversal graf.