

LAPORAN PRAKTIKUM
Modul 14
“Graph”



Disusun Oleh:
Ganesha Rahman Gibran -2211104058
Kelas S1SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

A. TUJUAN PRAKTIKUM

- a. Memahami konsep graph.
- b. Mengimplementasikan graph dengan menggunakan pointer

B. DASAR TEORI

Graph

Graph adalah himpunan tidak kosong dari simpul (node/verteks) dan garis penghubung (edge). Contoh sederhana penggunaan graph adalah antara Tempat Kost Anda dan Common Lab. Tempat Kost Anda dan Common Lab merupakan simpul (node/verteks), sedangkan jalan yang menghubungkan keduanya merupakan garis penghubung (edge).

Jenis-jenis Graph

1. Graph Berarah

Graph berarah adalah graph di mana setiap simpul memiliki garis penghubung (edge) yang menunjukkan arah keterhubungan simpul tersebut.

Representasi Graph Berarah

Representasi graph berarah secara umum hampir sama dengan graph tak-berarah. Perbedaannya adalah pada graph tak-berarah, jika simpul A terhubung dengan simpul B, maka ada panah bolak-balik dari A ke B dan dari B ke A. Sedangkan pada graph berarah, jika simpul A terhubung ke simpul B, belum tentu B terhubung ke A. Graph berarah dapat direpresentasikan dalam bentuk multilist karena sifat list yang dinamis

2. Graph Tak Berarah

Graph tidak berarah adalah graph di mana setiap simpul memiliki garis penghubung yang tidak memiliki arah tertentu. Ketetanggaan antar simpul harus diperhatikan dalam implementasi program karena digunakan untuk beberapa proses seperti pencarian lintasan terpendek, konversi graph menjadi tree, dan lainnya.

GUIDED 1

Sourcecode

```
#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;
}

```

Output

```

Graph\Guided\Output
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\14_Graph\Guided\output> & .
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\14_Graph\Guided\output>

```

C. UNGUIDED

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

Output Program

```

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

      BALI    PALU
BALI   0      3
PALU   4      0

Process returned 0 (0x0)   execution time : 11.763 s
Press any key to continue.

```

Sourcecode

```

#include <iostream>
#include <vector>
#include <string>
#include <iomanip> // Untuk format output

using namespace std;

int main() {
    int jumlahSimpul;

    // Meminta jumlah simpul dari user
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jumlahSimpul;

    vector<string> namaSimpul(jumlahSimpul);
    vector<vector<int>> graph(jumlahSimpul, vector<int>(jumlahSimpul, 0));

    // Input nama simpul
    for (int i = 0; i < jumlahSimpul; i++) {

```

```

        cout << "Silakan masukkan nama simpul " << i + 1 << ": ";
        cin >> namaSimpul[i];
    }

    // Input bobot antar simpul
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << namaSimpul[i] << "--> " << namaSimpul[j] << " = ";
            cin >> graph[i][j];
        }
    }

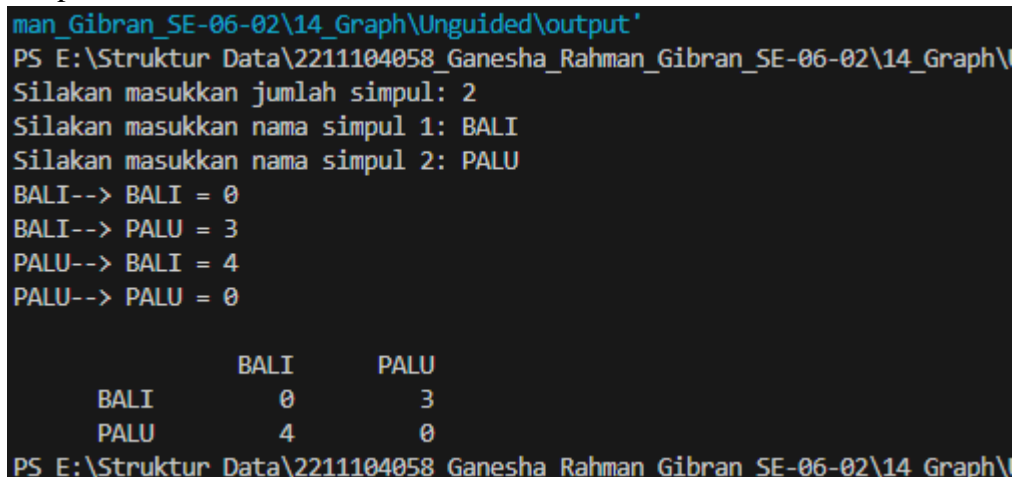
    // Output matriks adjacency
    cout << endl << setw(10) << "";
    for (const auto& nama : namaSimpul) {
        cout << setw(10) << nama;
    }
    cout << endl;

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

    return 0;
}

```

Output



```

man_Gibran_SE-06-02\14_Graph\Unguided\output'
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\14_Graph\
Silakan masukkan jumlah simpul: 2
Silakan masukkan nama simpul 1: BALI
Silakan masukkan nama simpul 2: PALU
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

      BALI    PALU
BALI    0      3
PALU    4      0
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\14_Graph\

```

2. Buatlah sebuah program C++ untuk merepresentasikan graf tidak berarah menggunakan adjacency matrix. Program harus dapat:
 - Menerima input jumlah simpul dan jumlah sisi.
 - Menerima input pasangan simpul yang terhubung oleh sisi.
 - Menampilkan adjacency matrix dari graf tersebut.

Input Contoh:

Masukkan jumlah simpul: 4

Masukkan jumlah sisi: 4

Masukkan pasangan simpul:

1 2

1 3

2 4

3 4

Output Contoh:

Adjacency Matrix:

0 1 1 0

1 0 0 1

1 0 0 1

0 1 1 0

Sourcecode :

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

int main() {
    int numVertices, numEdges;

    // Memasukkan 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));

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

        // Karena graf tidak berarah, tandai kedua arah dalam adjacency matrix
        adjMatrix[u - 1][v - 1] = 1;
        adjMatrix[v - 1][u - 1] = 1;
    }

    // Menampilkan adjacency matrix
    cout << "Adjacency Matrix:" << endl;
    for (int i = 0; i < numVertices; ++i) {
        for (int j = 0; j < numVertices; ++j) {
            cout << adjMatrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output :

```
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\14_Graph\Unguided\output>
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul:
1 2
1 3
2 4
3 4
Adjacency Matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02\14_Graph\Unguided\output>
```

D. KESIMPULAN

Implementasi graph menggunakan C++ dengan metode adjacency list memberikan representasi yang efisien untuk menyimpan dan memanipulasi data struktur graf. Fitur utama meliputi penambahan simpul (node), garis penghubung (edge), pencetakan graf, dan topological sort untuk menentukan urutan eksekusi simpul tanpa siklus. Program ini memastikan integritas struktur graf dengan memeriksa siklus sebelum melakukan topological sort. Dengan modularitas yang baik melalui file header dan implementasi terpisah, solusi ini cocok untuk berbagai aplikasi seperti pemodelan dependensi, jadwal tugas, atau analisis jaringan.