

LAPORAN PRAKTIKUM
Modul 14
GRAPH



Disusun Oleh:
Aulia Jasifa Br Ginting 2311104060
S1SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

1. Memahami konsep graph.
2. Mengimplementasikan graph dengan menggunakan pointer.

2. Landasan Teori

Graph

Graph adalah struktur data yang terdiri dari kumpulan simpul (vertices) dan sisi (edges) yang menghubungkan antar simpul tersebut. Graph dapat digunakan untuk merepresentasikan objek-objek diskrit dan hubungan di antara mereka.

- **Graph Berarah (Directed Graph):**

Adalah jenis graph di mana setiap sisi (edge) memiliki arah tertentu. Dalam graph berarah, hubungan antara simpul (vertices) bersifat satu arah, yang berarti bahwa jika ada sisi dari simpul A ke simpul B, maka hanya A yang dapat mengarah ke B, tetapi tidak sebaliknya, kecuali ada sisi terpisah dari B ke A. Contoh: graf yang merepresentasikan hubungan satu arah, seperti pengikut di media sosial.

- **Graph Tidak Berarah (Undirected Graph):**

Adalah jenis graph di mana sisi (edge) yang menghubungkan dua simpul (vertices) tidak memiliki arah. Ini berarti bahwa jika ada sisi yang menghubungkan simpul A dan simpul B, maka hubungan tersebut bersifat dua arah; artinya, kita dapat bergerak dari A ke B dan juga dari B ke A. Contoh: graf yang merepresentasikan hubungan pertemanan di mana kedua pengguna saling terhubung.

3. Guided

Code programnya:

```
#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;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
```

4. Unguided

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

Code programnya

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

class Graph {
private:
    int vertices;
    vector<vector<int>> adjacencyMatrix;
    vector<string> cityNames;

public:
    // Constructor
    Graph(int v) {
        vertices = v;
        adjacencyMatrix.resize(v, vector<int>(v, 0));
        cityNames.resize(v);
    }

    // Menambahkan nama kota
    void addCityName(int index, string name) {
        cityNames[index] = name;
    }

    // Menambahkan edge dengan bobot
    void addEdge(int from, int to, int weight) {
        adjacencyMatrix[from][to] = weight;
    }
}
```

```
}

// Menampilkan matrix adjacency
void displayMatrix() {
    // Menampilkan header kolom
    cout << "\n\t";
    for(int i = 0; i < vertices; i++) {
        cout << cityNames[i] << "\t";
    }
    cout << endl;

    // Menampilkan matrix
    for(int i = 0; i < vertices; i++) {
        cout << cityNames[i] << "\t";
        for(int j = 0; j < vertices; j++) {
            cout << adjacencyMatrix[i][j] << "\t";
        }
        cout << endl;
    }
}

// Getter untuk nama kota
string getCityName(int index) {
    return cityNames[index];
}
};

int main() {
    int numVertices;

    cout << "Silakan masukan jumlah simpul : ";
    cin >> numVertices;

    Graph graph(numVertices);

    cout << "Silakan masukan nama simpul" << endl;
    for(int i = 0; i < numVertices; i++) {
        string cityName;
        cout << "Simpul " << i + 1 << " : ";
        cin >> cityName;
        graph.addCityName(i, cityName);
    }

    cout << "Silakan masukkan bobot antar simpul" << endl;
    for(int i = 0; i < numVertices; i++) {
        for(int j = 0; j < numVertices; j++) {
            if(i != j) {
                int weight;
```

```
        string fromCity = graph.getCityName(i);
        string toCity = graph.getCityName(j);
        cout << fromCity << "---> " << toCity << " = ";
        cin >> weight;
        graph.addEdge(i, j, weight);
    }
}

// Menampilkan hasil matrix adjacency
graph.displayMatrix();

return 0;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\
Silakan masukan jumlah simpul : 2
Silakan masukan nama simpul
Simpul 1 : BALI
Simpul 2 : PALU
Silakan masukkan bobot antar simpul
BALI---> PALU = 4
PALU---> BALI = 3

      BALI    PALU
BALI   0      4
PALU   3      0
```

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.

Code programnya

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

int main() {
    int jumlahSimpul, jumlahSisi;

    // Input jumlah simpul dan sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan jumlah sisi: ";
    cin >> jumlahSisi;

    // Inisialisasi adjacency matrix dengan 0
```



```
int adjacencyMatrix[10][10] = {0}; // Asumsi maksimal 10 simpul

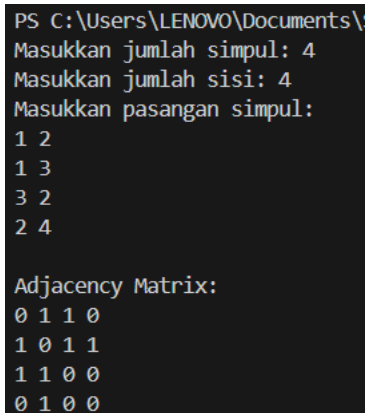
cout << "Masukkan pasangan simpul:\n";
for(int i = 0; i < jumlahSisi; i++) {
    int simpul1, simpul2;
    cin >> simpul1 >> simpul2;

    // Karena graf tidak berarah, kita set kedua arah
    adjacencyMatrix[simpul1-1][simpul2-1] = 1;
    adjacencyMatrix[simpul2-1][simpul1-1] = 1;
}

// Menampilkan adjacency matrix
cout << "\nAdjacency Matrix:\n";
for(int i = 0; i < jumlahSimpul; i++) {
    for(int j = 0; j < jumlahSimpul; j++) {
        cout << adjacencyMatrix[i][j] << " ";
    }
    cout << endl;
}

return 0;
}
```

Outputnya:



```
PS C:\Users\LENOVO\Documents\  
Masukkan jumlah simpul: 4  
Masukkan jumlah sisi: 4  
Masukkan pasangan simpul:  
1 2  
1 3  
3 2  
2 4  
  
Adjacency Matrix:  
0 1 1 0  
1 0 1 1  
1 1 0 0  
0 1 0 0
```

5. Kesimpulan

Pada praktikum ini, mempelajari konsep dasar graph sebagai struktur data yang terdiri dari simpul (vertices) dan sisi (edges) yang menghubungkan antar simpul. membedakan antara graph berarah (directed graph) dan graph tidak berarah (undirected graph), di mana graph berarah memiliki hubungan satu arah, sedangkan graph tidak



berarah memiliki hubungan dua arah. dan mengimplementasikan graph menggunakan pointer, yang memungkinkan pengelolaan memori yang efisien dan fleksibilitas dalam menyimpan hubungan antar simpul.