LAPORAN PRAKTIKUM

MODUL 14

“GRAPH”

Disusun Oleh:

Dhiemas Tulus Ikhsan 2311104046

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. TUJUAN

a. Mamahami konsep graph

b. Mengimplementasikan graph dengan menggunakan pointer

II. LANDASAN TEORI

Graph adalah salah satu struktur data yang terdiri dari kumpulan simpul (node atau vertex) dan garis penghubung antar simpul tersebut (edge). Struktur ini sering digunakan untuk merepresentasikan berbagai hubungan dalam kehidupan nyata, seperti jaringan sosial, peta, atau hubungan dalam sistem komputer. Berdasarkan karakteristiknya, graph dapat dibagi menjadi dua jenis, yaitu graph berarah dan graph tidak berarah.

Graph berarah adalah graph di mana setiap edge memiliki arah tertentu yang menunjukkan hubungan dari satu simpul ke simpul lainnya. Contoh penerapan graph berarah dapat ditemukan pada sistem transportasi dengan rute satu arah, di mana perjalanan hanya memungkinkan ke arah yang telah ditentukan. Di sisi lain, graph tidak berarah merupakan jenis graph yang edge-nya tidak memiliki arah tertentu, sehingga hubungan antara dua simpul bersifat timbal balik. Contoh penerapannya dapat dilihat pada hubungan pertemanan di media sosial, di mana koneksi antara dua individu bersifat dua arah.

Dalam proses eksplorasi atau penelusuran graph, terdapat dua metode utama yang sering digunakan, yaitu Breadth First Search (BFS) dan Depth First Search (DFS). BFS adalah metode yang menjelajahi graph secara berlapis, dimulai dari simpul awal (root), kemudian bergerak ke semua simpul pada level yang sama sebelum melanjutkan ke level berikutnya. Metode ini cocok untuk mencari jalur terpendek dalam graf tak berberat. Sebaliknya, DFS adalah metode yang menjelajahi graph secara mendalam dengan menyusuri satu cabang hingga simpul terjauh sebelum kembali ke simpul sebelumnya untuk melanjutkan ke cabang lainnya. Metode ini sering digunakan untuk mendeteksi siklus atau melakukan traversal seluruh simpul pada graph.

Dengan kedua metode tersebut, penelusuran graph menjadi lebih terstruktur dan efisien, tergantung pada kebutuhan aplikasi yang dihadapi. Graph, sebagai representasi hubungan yang kompleks, memiliki peran penting dalam menyelesaikan berbagai permasalahan dunia nyata.

III. GUIDED

1. guided

```
#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;
};

// Fungsi untuk membuat graf baru
void CreateGraph(Graph &G) {
    G.first = NULL; // Inisialisasi graf kosong
}

// Fungsi untuk menambahkan node baru ke graf
void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode; // Alokasi memori untuk node baru
    newNode->info = X;              // Menyimpan informasi
    newNode->visited = false;       // Status awal belum dikunjungi
    newNode->firstEdge = NULL;      // Belum ada tepi
    newNode->Next = NULL;           // Node berikutnya adalah NULL

    if (G.first == NULL) {
        G.first = newNode; // Jika graf kosong, tambahkan sebagai simpul pertama
    } else {
        ElmNode *temp = G.first;
        while (temp->Next != NULL) {
            temp = temp->Next; // Cari node terakhir dalam daftar
        }
        temp->Next = newNode; // Tambahkan node baru di akhir
    }
}

// Fungsi untuk membuat hubungan antar node (menambahkan tepi)
void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge; // Alokasi memori untuk tepi baru
    newEdge->Node = N2;             // Tepi menghubungkan ke simpul N2
    newEdge->Next = N1->firstEdge; // Sisipkan di awal daftar tepi
    N1->firstEdge = newEdge;        // Perbarui daftar tepi N1
}

// Fungsi untuk mencetak informasi node dalam graf
void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " "; // Cetak informasi setiap simpul
        temp = temp->Next;         // Lanjut ke simpul berikutnya
    }
    cout << endl;
}

// Fungsi untuk mengatur ulang status visited semua simpul
void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false; // Reset visited ke false
        temp = temp->Next;
    }
}

// Fungsi untuk traversal graf menggunakan DFS
void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return; // Basis rekursi, jika simpul NULL, selesai
    }
    N->visited = true; // Tandai simpul telah dikunjungi
```

```
    cout << N->info << " "; // Cetak informasi simpul

    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node); // Rekursi ke simpul yang terhubung
        }
        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:

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
```

IV. UNGUIDED

1. Task_1

```
#include <iostream>
#include <string>
#include <iomanip>
#define MAX_VERTICES 100
using namespace std;

int main() {
    int totalVertices;
    string vertexNames[MAX_VERTICES];
    int weightMatrix[MAX_VERTICES][MAX_VERTICES];

    // Memasukkan jumlah simpul
    cout << "Masukkan jumlah simpul: ";
    cin >> totalVertices;

    // Memasukkan nama-nama simpul
    for (int idx = 0; idx < totalVertices; ++idx) {
        cout << "Nama simpul " << idx + 1 << ": ";
        cin >> vertexNames[idx];
    }

    // Memasukkan bobot antar simpul dengan tabel
    cout << "\nMasukkan bobot antar simpul dalam bentuk matriks:\n";
    for (int row = 0; row < totalVertices; ++row) {
        for (int col = 0; col < totalVertices; ++col) {
            cout << "Bobot " << vertexNames[row] << " ke " << vertexNames[col] << ":
";
            cin >> weightMatrix[row][col];
        }
    }

    // Menampilkan matriks bobot
    cout << "\nMatriks Bobot:\n";
    cout << setw(10) << " ";
    for (int col = 0; col < totalVertices; ++col) {
        cout << setw(10) << vertexNames[col];
    }
    cout << "\n";

    for (int row = 0; row < totalVertices; ++row) {
        cout << setw(10) << vertexNames[row];
        for (int col = 0; col < totalVertices; ++col) {
            cout << setw(10) << weightMatrix[row][col];
        }
        cout << "\n";
    }

    return 0;
```

```
}
```

Output:

```
Masukkan jumlah simpul: 2
Nama simpul 1: look
Nama simpul 2: kool

Masukkan bobot antar simpul dalam bentuk matriks:
Bobot look ke look: 3
Bobot look ke kool: 4
Bobot kool ke look: 5
Bobot kool ke kool: 6

Matriks Bobot:
               look      kool
      look        3         4
      kool        5         6
```

2. Task_2

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

int main() {
    int numVertices, numEdges;

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

    // Deklarasi adjacency matrix
    vector<vector<int>>                          adjacencyMatrix(numVertices,
   vector<int>(numVertices, 0));
    vector<string> vertexLabels(numVertices);

    // Input label simpul
    cout << "Masukkan nama simpul:\n";
    for (int i = 0; i < numVertices; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> vertexLabels[i];
    }

    // Input sisi antar simpul
    cout << "\nMasukkan pasangan simpul yang terhubung:\n";
    for (int i = 0; i < numEdges; ++i) {
        string from, to;
        cout << "Sisi " << i + 1 << ": ";
        cin >> from >> to;

        int fromIdx = -1, toIdx = -1;
        for (int j = 0; j < numVertices; ++j) {
            if (vertexLabels[j] == from) fromIdx = j;
            if (vertexLabels[j] == to) toIdx = j;
        }
```

```
        if (fromIdx != -1 && toIdx != -1) {
            adjacencyMatrix[fromIdx][toIdx] = 1;
            adjacencyMatrix[toIdx][fromIdx] = 1; // Untuk graf tak berarah
        } else {
            cout << "Nama simpul tidak valid. Coba lagi.\n";
            --i;
        }
    }

    // Menampilkan adjacency matrix
    cout << "\nAdjacency Matrix:\n";
    cout << " ";
    for (const auto& label : vertexLabels) {
        cout << "\t" << label;
    }
    cout << "\n";

    for (int i = 0; i < numVertices; ++i) {
        cout << vertexLabels[i];
        for (int j = 0; j < numVertices; ++j) {
            cout << "\t" << adjacencyMatrix[i][j];
        }
        cout << "\n";
    }

    return 0;
}
```

Output:

```
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 3
Masukkan nama simpul:
Simpul 1: A
Simpul 2: B
Simpul 3: C
Simpul 4: D

Masukkan pasangan simpul yang terhubung:
Sisi 1: A
D
Sisi 2: B
B
Sisi 3: C
C

Adjacency Matrix:
        A       B       C       D
A       0       0       0       1
B       0       1       0       0
C       0       0       1       0
D       1       0       0       0
```

V. KESIMPULAN

Pada praktikum ini, telah berhasil diimplementasikan representasi graf menggunakan adjacency matrix. Melalui program yang dibuat, simpul dan hubungan antar simpul dapat diinput secara fleksibel, baik dari jumlah simpul maupun pasangan simpul yang terhubung. Output berupa adjacency matrix menunjukkan representasi graf dalam bentuk matriks biner, yang memudahkan visualisasi dan analisis hubungan antar simpul.

Praktikum ini menunjukkan bahwa adjacency matrix adalah metode representasi graf yang efisien untuk graf dengan jumlah simpul yang relatif kecil dan dapat diimplementasikan secara efektif menggunakan struktur data array dua dimensi.