LAPORAN PRAKTIKUM

Modul 14

GRAF

Disusun Oleh:

Adhiansyah Muhammad Pradana Farawowan - 2211104038

S1SE-07-02

Asisten Praktikum:

Aldi Putra

Andini Nur Hidayah

Dosen:

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

PROGRAM STUDI S1 REKAYASAN PERANGKAT LUNAK

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM PURWOKERTO

2024

A. Tujuan

Laporan praktikum ini memiliki tujuan di bawah berikut.

1. Memperkenalkan konsep graf sebagai salah satu struktur data
2. Memahami dan mengelola cara kerja graf
3. Mengimplementasikan graf dalam bahasa C++

B. Landasan Teori

Graf adalah struktur data yang berisi kumpulan simpul (*vertex*) yang terhubung oleh kumpulan sisi (*edge*).

C. Bimbingan (*guided*)

Bimbingan hari ini adalah mengimplementasikan sebuah graf dengan modifikasi sendiri.

guided.cpp

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

struct ElmNode;

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

struct ElmNode
{
    char info;
    bool visited;
    ElmEdge *first_edge;
    ElmNode *next;
};

struct Graph
{
    ElmNode *first;
};

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

void insert_node(Graph &G, char X)
{
    ElmNode *new_node = new ElmNode;
    new_node->info = X;
    new_node->visited = false;
    new_node->first_edge = NULL;
    new_node->next = NULL;

    if (G.first == NULL)
    {
        G.first = new_node;
    }
    else
    {
        ElmNode *temp = G.first;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = new_node;
    }
```

```
}

void connect_node(ElmNode *N1, ElmNode *N2)
{
    ElmEdge *new_edge = new ElmEdge;
    new_edge->Node = N2;
    new_edge->next = N1->first_edge;
    N1->first_edge = new_edge;
}

void print_infos_graph(Graph G)
{
    ElmNode *temp = G.first;
    while (temp != NULL)
    {
        std::cout << temp->info << " ";
        temp = temp->next;
    }
    std::cout << '\n';
}

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

void traverse_dfs(Graph G, ElmNode *N)
{
    if (N == NULL)
    {
        return;
    }
    N->visited = true;
    std::cout << N->info << " ";
    ElmEdge *edge = N->first_edge;
    while (edge != NULL)
    {
        if (!edge->Node->visited)
        {
            traverse_dfs(G, edge->Node);
        }
        edge = edge->next;
    }
}

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

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

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

```
int main()
{
    Graph G;
    create_graph(G);

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

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

    std::cout << "DFS traversal: ";
    reset_visited(G);
    traverse_dfs(G, A);
    std::cout << '\n';

    std::cout << "BFS traversal: ";
    reset_visited(G);
    traverse_bfs(G, A);
    std::cout << '\n';

    return 0;
}
```

Output dari guided_1.cpp

```
                                                    >a.exe
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
```

D. Tugas mandiri (*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.
```

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
   menggunakan adjacency matrix. Program harus dapat:
   Output Contoh:
   Adjacency Matrix:
   0 1 1 0
   1 0 0 1
   1 0 0 1
   0 1 1 0

Kode

unguided_1.cpp

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

struct Vertex
{
    std::string name;
    std::map<std::string, int> weighted_edges;
};

struct Graph
{
    // Vektor pada dasarnya adalah array yang dapat berubah ukuran
    // Vektor tanpa menetapkan ukuran awal akan mulai dari ukuran 0 (nol)
    std::vector<Vertex> vertices;
};

int main()
{
```

```
    int n;
    Graph cities;

    std::string input_n;

    std::cout << "Masukkan jumlah simpul yang ingin dibuat: ";
    std::cin >> n;

    std::cout << "Masukkan nama simpul:\n";

    while (cities.vertices.size() != n)
    {
        std::cout << "Simpul " << cities.vertices.size() + 1 << ": ";
        std::cin >> input_n;

        Vertex v;
        v.name = input_n;

        cities.vertices.push_back(v);
    }

    std::cout << "Masukkan jarak kota: " << '\n';

    int distance;
    for (int i = 0; i < n; i = i + 1)
    {
        for (int j = 0; j < n; j = j + 1)
        {
            if (i != j)
            {
                std::cout << cities.vertices[i].name << " ---> " << cities.vertices[j].name
<< ": ";
                std::cin >> distance;

                cities.vertices[i].weighted_edges[cities.vertices[j].name] = distance;
            }
        }
    }

    std::cout << '\n';

    std::cout << "Matriks tetangga:\n";
    std::cout << "   ";
    for (int _i = 0; _i < n; _i = _i + 1)
    {
        std::cout << "(" << _i + 1 << ")" << " ";
    }

    std::cout << '\n';

    for (int i = 0; i < n; i = i + 1)
    {
        std::cout << "(" << i + 1 << ")" << " ";
        for (int j = 0; j < n; j = j + 1)
        {
            std::cout << cities.vertices[i].weighted_edges[cities.vertices[j].name] << " ";
        }

        std::cout << '\n';
    }

    return 0;
}
```

Output dari guided_1.cpp

```
>a.exe
Masukkan jumlah simpul yang ingin dibuat: 2
Masukkan nama simpul:
Simpul 1: Purwokerto
Simpul 2: Semarang
Masukkan jarak kota:
Purwokerto ---> Semarang: 4
Semarang ---> Purwokerto: 5

Matriks tetangga:
   (1) (2)
(1) 0 4
(2) 5 0
```

unguided_2.cpp

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

struct Vertex
{
    std::string name;
    std::vector<std::string> edges_to_vertices;
};

struct Graph
{
    std::vector<Vertex> vertices;
};

// Pada dasarnya, kita buat grapf biasa
int main()
{
    Graph normal_graph;

    int n_vertices;
    int n_edges;

    std::cout << "Masukkan jumlah simpul: ";
    std::cin >> n_vertices;

    std::cout << "Masukkan jumlah sisi: ";
    std::cin >> n_edges;

    std::cout << "Isikan pasangan yang diinginkan (seperti: 1 2, simpul 1 ke simpul 2): "
<< '\n';

    std::string v1_name;
    std::string v2_name;
    for (int i = 0; i < n_edges; i = i + 1)
    {
        std::cin >> v1_name >> v2_name;

        Vertex v1;
        Vertex v2;
        v1.name = v1_name;
        v2.name = v2_name;

        for (int x = 0; x < normal_graph.vertices.size(); x = x + 1)
        {
            if (normal_graph.vertices[x].name == v1_name || normal_graph.vertices[x].name
== v1_name)
            {
                normal_graph.vertices[x].edges_to_vertices.push_back(v2_name);
                break;
            }
```

```
        }

        normal_graph.vertices.push_back(v1);
        normal_graph.vertices.push_back(v2);
    }

    return 0;
}
```

Output dari guided_2.cpp

```
>a.exe
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Isikan pasangan yang diinginkan (seperti: 1 2, simpul 1 ke simpul 2):
a b
a c
c d
b d
```