LAPORAN PRAKTIKUM
Modul 13
“GRAPH”

Disusun Oleh:
Aji Prasetyo Nugroho - 2211104049
S1SE-07-2

Assisten Praktikum :
Aldi Putra
Andini Nur Hidayah

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

A. Tujuan

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

B. Landasan Teori

1. Pengertian Graph

   Graph merupakan salah satu struktur data dalam bidang ilmu komputer dan matematika diskret yang terdiri dari kumpulan simpul (node atau vertex) dan sisi (edge) yang menghubungkan simpul-simpul tersebut. Graph digunakan untuk merepresentasikan hubungan atau koneksi antar objek. Secara formal, graph dapat dinyatakan sebagai pasangan , di mana:

   - adalah himpunan tidak kosong dari simpul.
   - adalah himpunan dari sisi yang menghubungkan dua simpul dalam .

2. Jenis-Jenis Graph

   1) **Berdasarkan Arah:**
      - **Graph Berarah (Directed Graph):** Graph di mana setiap sisi memiliki arah tertentu, dinotasikan sebagai pasangan berurut .
      - **Graph Tak Berarah (Undirected Graph):** Graph di mana setiap sisi tidak memiliki arah, dinotasikan sebagai pasangan tidak berurut {u, v}.
   2) **Berdasarkan Bobot:**
      - **Graph Berbobot (Weighted Graph):** Graph di mana setiap sisi memiliki nilai atau bobot tertentu.
      - **Graph Tak Berbobot (Unweighted Graph):** Graph di mana sisi tidak memiliki bobot.
   3) **Berdasarkan Hubungan Simpul:**
      - **Graph Terhubung (Connected Graph):** Graph di mana setiap simpul memiliki jalur ke simpul lainnya.
      - **Graph Tak Terhubung (Disconnected Graph):** Graph di mana terdapat simpul yang tidak memiliki jalur ke simpul lainnya.
   4) **Berdasarkan Struktur:**
      - **Graph Siklik (Cyclic Graph):** Graph yang memiliki lintasan melingkar.

- **Graph Asiklik (Acyclic Graph):** Graph yang tidak memiliki lintasan melingkar.

3. Representasi Graph

Untuk merepresentasikan graph dalam implementasi, ada beberapa cara yang umum digunakan:

- **Matriks Ketetanggaan (Adjacency Matrix):** Representasi menggunakan matriks , di mana adalah jumlah simpul. Elemen matriks bernilai 1 jika terdapat sisi antara simpul dan , dan 0 jika tidak.
- **Daftar Ketetanggaan (Adjacency List):** Representasi menggunakan daftar, di mana setiap simpul memiliki daftar simpul tetangga yang terhubung dengannya.
- **Matriks Insidensi (Incidence Matrix):** Representasi menggunakan matriks , di mana adalah jumlah simpul dan adalah jumlah sisi. Elemen matriks menunjukkan hubungan antara simpul dan sisi.

4. Aplikasi Graph

Graph memiliki aplikasi luas dalam berbagai bidang, antara lain:

1) **Ilmu Komputer:**
   - Algoritma pencarian jalur terpendek (Dijkstra, A*).
   - Representasi jaringan komputer.
   - Analisis media sosial (graf koneksi antar pengguna).
2) **Ilmu Transportasi:**
   - Representasi jalur transportasi.
   - Optimasi rute (Traveling Salesman Problem, Vehicle Routing Problem).
3) **Matematika dan Fisika:**
   - Representasi jaringan listrik.
   - Model dinamika fluida pada jaringan.
4) **Biologi:**
   - Representasi jaringan genetik.
   - Analisis hubungan ekosistem.
5) **Bisnis dan Ekonomi:**
   - Analisis rantai suplai.
   - Model jaringan kerja.

5. Algoritma pada Graph

Beberapa algoritma penting yang sering digunakan dalam operasi pada graph adalah:

1. **Pencarian dalam Graph:**
   - Depth First Search (DFS)
   - Breadth First Search (BFS)
2. **Pencarian Jalur Terpendek:**
   - Algoritma Dijkstra
   - Algoritma Bellman-Ford
3. **Minimum Spanning Tree:**
   - Algoritma Kruskal
   - Algoritma Prim
4. **Deteksi Siklus:**
   - Algoritma Union-Find

C. Guided

- Guided 1

  Source code :

```
#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 :

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
PS D:\Praktikum STD_2211104049>
```

D. Unguided

Latihan soal

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
   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

- **Unguided 1**

  Source Code :

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

int main() {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
```

```
    cin >> n;

    vector<string> vertices(n);
    cout << "Silakan masukan nama simpul" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> vertices[i];
    }

    vector<vector<int>> graph(n, vector<int>(n));
    cout << "\nSilakan masukan bobot antar simpul" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << vertices[i] << " --> " << vertices[j] << ": ";
            cin >> graph[i][j];
        }
    }

    cout << "\n\t";
    for (const auto& vertex : vertices) {
        cout << vertex << "\t";
    }
    cout << endl;

    for (int i = 0; i < n; i++) {
        cout << vertices[i] << "\t";
        for (int j = 0; j < n; j++) {
            cout << graph[i][j] << "\t";
        }
        cout << endl;
    }

    return 0;
}
```

Output :

```
Silakan masukan jumlah simpul: 3
Silakan masukan nama simpul
Simpul 1: Aji
Simpul 2: Prasetyo
Simpul 3: Nugroho

Silakan masukan bobot antar simpul
Aji --> Aji: 2
Aji --> Prasetyo: 4
Aji --> Nugroho: 6
Prasetyo --> Aji: 7
Prasetyo --> Prasetyo: 1
Prasetyo --> Nugroho: 9
Nugroho --> Aji: 2
Nugroho --> Prasetyo: 4
Nugroho --> Nugroho: 1

          Aji     Prasetyo        Nugroho
Aji       2       4       6
Prasetyo        7       1       9
Nugroho 2       4       1
PS D:\Praktikum STD_2211104049>
```

- **Unguided 2**

  Source code :

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

int main() {
    int vertices, edges;

    // Input jumlah simpul
    cout << "Masukkan jumlah simpul: ";
    cin >> vertices;

    // Input jumlah sisi
    cout << "Masukkan jumlah sisi: ";
    cin >> edges;

    // Inisialisasi matrix adjacency dengan 0
    vector<vector<int>> adjMatrix(vertices, vector<int>(vertices, 0));
```

```
    cout << "Masukkan pasangan simpul:\n";
    for(int i = 0; i < edges; i++) {
        int v1, v2;
        cin >> v1 >> v2;
        // Karena graf tidak berarah, set kedua arah
        adjMatrix[v1-1][v2-1] = 1;
        adjMatrix[v2-1][v1-1] = 1;
    }

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

    return 0;
}
```

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 D:\Praktikum STD_2211104049>
```