

**LAPORAN PRAKTIKUM
STRUKTUR DATA
Modul 14
“GRAPH”**



Disusun Oleh:
MUHAMMAD RALFI - 2211104054
SE-07-2

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024**

1. Tujuan

- Mahasiswa dapat memahami konsep Graph
- Mahasiswa mampu mendefinisikann tentang Graph pada pemrograman
- Mahasiswa dapat mengimplementasikan konsep Graph pada pemrograman

2. Landasan Teori

Graph

Graf (Graph) merupakan struktur data non-linear yang terdiri dari sekumpulan vertex (node/simpul) yang saling terhubung melalui edge (sisi/busur), dimana setiap edge menghubungkan tepat dua buah vertex. Struktur data graf memungkinkan representasi hubungan yang lebih kompleks antara objek-objek dalam suatu sistem, seperti jaringan sosial, peta, atau sistem navigasi. Graf dapat dikategorikan menjadi beberapa jenis berdasarkan karakteristiknya, seperti graf berarah (directed graph) dimana edge memiliki arah tertentu, graf tidak berarah (undirected graph) dimana edge tidak memiliki arah, graf berbobot (weighted graph) dimana setiap edge memiliki nilai atau bobot tertentu.

Dalam implementasinya, graf dapat direpresentasikan menggunakan beberapa metode seperti matriks ketetanggaan (adjacency matrix) dan senarai ketetanggaan (adjacency list). Representasi menggunakan adjacency matrix cocok untuk graf yang memiliki banyak edge (dense graph), sedangkan adjacency list lebih efisien untuk graf dengan edge yang sedikit (sparse graph). Graf juga memiliki berbagai algoritma traversal seperti Depth First Search (DFS) dan Breadth First Search (BFS) yang digunakan untuk mengunjungi setiap vertex dalam graf, serta algoritma pencarian jalur terpendek seperti algoritma Dijkstra dan Floyd-Warshall yang sering digunakan dalam berbagai aplikasi seperti sistem GPS, jaringan komputer, dan optimasi rute pengiriman.

3. Guided

- Guided

File 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);

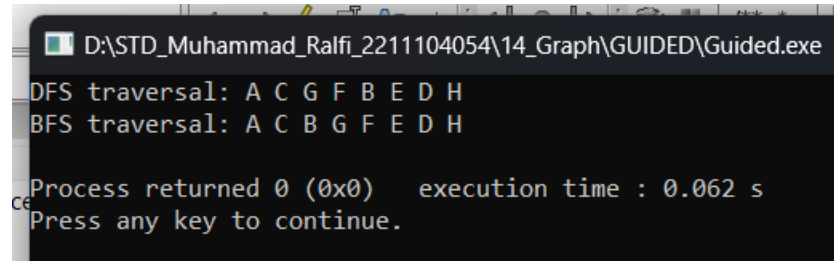
    cout << "DFS traversal: ";
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;

    cout << "BFS traversal: ";
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}

```

Output



```

D:\STD_Muhammad_Ralfi_2211104054\14_Graph\GUIDED\Guided.exe
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.

```

4. Unguided

a. Unguided 1

File Code

```

#include <iostream>
#include <vector>
#include <limits.h>

using namespace std;

void displayMatrix(const vector<vector<int>>& graph, const
vector<string>& cities) {

```

```
int n = graph.size();
cout << "\nAdjacency Matrix:\n";
cout << "      ";
for (const string& city : cities) {
    cout << city << " ";
}
cout << endl;
for (int i = 0; i < n; i++) {
    cout << cities[i] << " ";
    for (int j = 0; j < n; j++) {
        cout << graph[i][j] << " ";
    }
    cout << endl;
}
}

int main() {
    int n;
    cout << "Masukkan jumlah simpul: ";
    cin >> n;

    vector<string> cities(n);
    cout << "Masukkan nama simpul:\n";
    for (int i = 0; i < n; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> cities[i];
    }

    vector<vector<int>> graph(n, vector<int>(n, INT_MAX));

    cout << "\nMasukkan bobot antar simpul (jika tidak ada hubungan,
    masukkan 0):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                graph[i][j] = 0; // Jarak ke diri sendiri adalah 0
            } else {
                cout << cities[i] << " --> " << cities[j] << ": ";
                int weight;
                cin >> weight;
                graph[i][j] = (weight == 0) ? INT_MAX : weight;
            }
        }
    }

    displayMatrix(graph, cities);

    while (true) {
        cout << "\nHitung jarak dari kota ke kota (masukkan -1 untuk
        keluar):\n";
```

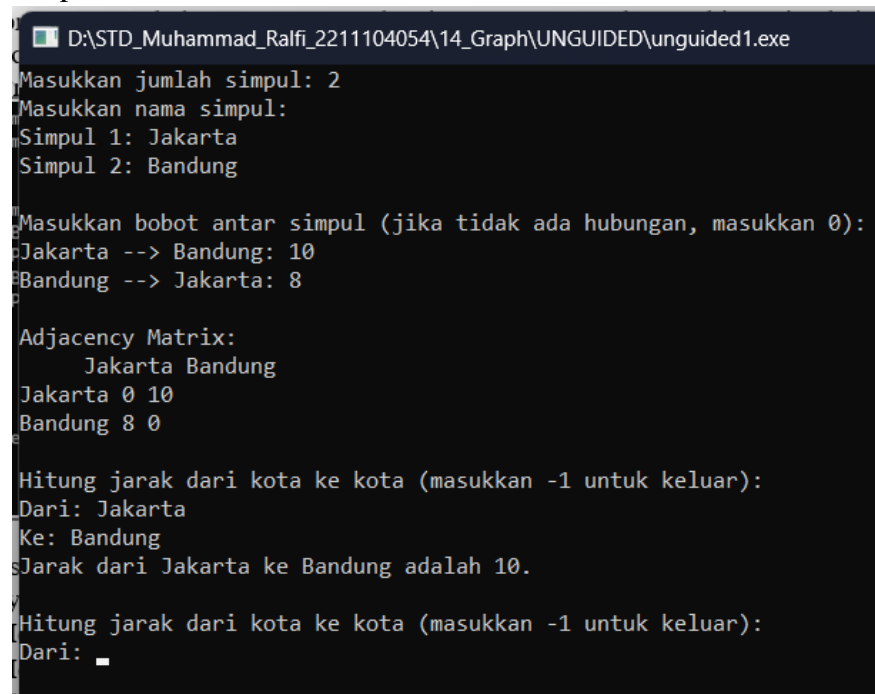
```
string from, to;
cout << "Dari: ";
cin >> from;
if (from == "-1") break;
cout << "Ke: ";
cin >> to;
if (to == "-1") break;

int fromIndex = -1, toIndex = -1;
for (int i = 0; i < n; i++) {
    if (cities[i] == from) fromIndex = i;
    if (cities[i] == to) toIndex = i;
}

if (fromIndex == -1 || toIndex == -1) {
    cout << "Kota tidak ditemukan. Coba lagi.\n";
} else if (graph[fromIndex][toIndex] == INT_MAX) {
    cout << "Tidak ada jalur langsung dari " << from << " ke "
<< to << ".\n";
} else {
    cout << "Jarak dari " << from << " ke " << to << " adalah "
<< graph[fromIndex][toIndex] << ".\n";
}

return 0;
}
```

Output



```
D:\STD_Muhammad_Ralfi_2211104054\14_Graph\UNGUIDED\unguided1.exe
Masukkan jumlah simpul: 2
Masukkan nama simpul:
Simpul 1: Jakarta
Simpul 2: Bandung

Masukkan bobot antar simpul (jika tidak ada hubungan, masukkan 0):
Jakarta --> Bandung: 10
Bandung --> Jakarta: 8

Adjacency Matrix:
    Jakarta Bandung
Jakarta 0 10
Bandung 8 0

Hitung jarak dari kota ke kota (masukkan -1 untuk keluar):
Dari: Jakarta
Ke: Bandung
Jarak dari Jakarta ke Bandung adalah 10.

Hitung jarak dari kota ke kota (masukkan -1 untuk keluar):
Dari: 
```

b. Unguided 2

File Code

```
#include <iostream>
#include <vector>

using namespace std;

void displayMatrix(const vector<vector<int>>& matrix) {
    int n = matrix.size();
    cout << "Adjacency Matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    int nodes, edges;

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

    // Inisialisasi adjacency matrix
    vector<vector<int>> adjMatrix(nodes, vector<int>(nodes, 0));

    // Input pasangan simpul yang terhubung
    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;

        // Karena graf tidak berarah, simetri adjacency matrix
        adjMatrix[u - 1][v - 1] = 1;
        adjMatrix[v - 1][u - 1] = 1;
    }

    // Tampilkan adjacency matrix
    displayMatrix(adjMatrix);

    return 0;
}
```


Output

```
D:\STD_Muhammad_Ralfi_2211104054\14_Graph\UNGUIDED\unguided
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

Process returned 0 (0x0)   execution time : 21.690 s
Press any key to continue.
```

5. Kesimpulan

Graf adalah salah satu struktur data yang digunakan untuk merepresentasikan hubungan antar elemen dalam bentuk simpul (nodes) dan sisi (edges). Dalam graf, simpul merepresentasikan entitas, sedangkan sisi merepresentasikan koneksi atau hubungan antara entitas tersebut. Graf dapat bersifat berarah (directed) atau tidak berarah (undirected), tergantung pada arah hubungan yang didefinisikan oleh sisi. Selain itu, graf juga dapat berbobot (weighted), di mana setiap sisi memiliki nilai tertentu, atau tidak berbobot (unweighted), di mana setiap sisi dianggap memiliki bobot yang sama.

Representasi graf secara umum dapat dilakukan menggunakan adjacency matrix atau adjacency list. Adjacency matrix menggunakan matriks dua dimensi untuk merepresentasikan hubungan antar simpul, di mana elemen matriks bernilai 1 jika ada hubungan dan 0 jika tidak ada. Representasi ini mudah dipahami dan digunakan untuk operasi seperti pengecekan cepat apakah dua simpul terhubung. Graf memiliki banyak aplikasi dalam berbagai bidang, seperti pemetaan jaringan, analisis hubungan sosial, dan pencarian jalur terpendek. Struktur ini sangat fleksibel dan dapat diperluas untuk menangani berbagai jenis masalah berbasis hubungan.

Output:

- a. Menambahkan node
- b. Mengecek BST apakah valid atau tidak
- c. Mencari simpul daun

6. Kesimpulan