LAPORAN PRAKTIKUM

Modul 14

Graph

Disusun Oleh :

Fauzan Rofif Ardiyanto/2211104036

SE 07 2

Asisten Praktikum :

Aldi Putra

Andini Nur Hidayah

Dosen Pengampu :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

1. Tujuan

a. Memahami konsep dasar graf (graph) sebagai himpunan simpul (nodes) dan sisi (edges).
b. Mempelajari representasi graf berarah dan tidak berarah menggunakan pointer.
c. Mengimplementasikan traversal graf dengan metode BFS dan DFS.
d. Membuat ADT graf untuk operasi dasar seperti membuat graf, menambah simpul, menghubungkan simpul, dan mencetak graf.
e. Menerapkan graf dalam kasus praktis seperti topological sorting untuk menyusun keterurutan data.

2. Landasan Teori

Graf adalah struktur data non-linear yang terdiri dari simpul (nodes) dan sisi (edges) yang menghubungkan simpul-simpul tersebut, digunakan untuk menggambarkan hubungan antar objek. Terdapat dua jenis graf utama, yaitu graf berarah (setiap sisi memiliki arah) dan graf tidak berarah (sisi tanpa arah tertentu). Graf banyak diterapkan dalam berbagai bidang, seperti jaringan komunikasi, peta jalan, dan analisis hubungan sosial.

Representasi graf dapat menggunakan matriks ketetanggaan atau daftar ketetanggaan, yang masing-masing memiliki kelebihan tergantung pada jumlah sisi dalam graf. Traversal graf dilakukan dengan algoritma Depth First Search (DFS) yang menjelajah secara mendalam, atau Breadth First Search (BFS) yang menjelajah secara melebar. Selain untuk traversal, graf juga digunakan dalam pengurutan topologis, yang menentukan urutan linier data dengan ketergantungan parsial. Dalam implementasi, graf sering menggunakan pointer dinamis untuk efisiensi penyimpanan dan manipulasi data yang lebih kompleks.

3. Guided

a. Guided 1

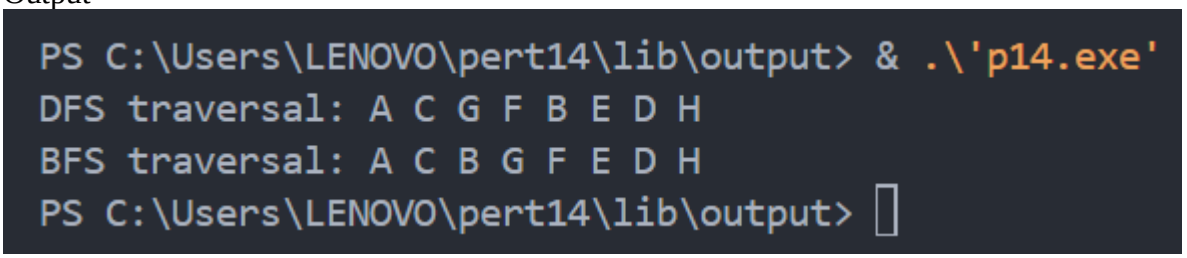
Source Code

```
#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



Deskripsi

Program C++ ini mengimplementasikan sebuah graf tak berarah menggunakan struktur data list terhubung (linked list). Program mencakup fungsi untuk membuat graf, menambahkan simpul (node), menghubungkan simpul dengan sisi (edge), dan melakukan traversal graf menggunakan metode DFS (Depth-First Search) dan BFS (Breadth-First Search). Graf terdiri dari simpul-simpul yang disimpan dalam struktur ElmNode, yang masing-masing memiliki informasi berupa karakter, status kunjungan (visited), dan daftar sisi yang terhubung. Metode DFS dan BFS digunakan untuk mengunjungi semua

simpul dalam graf, dengan DFS mengunjungi simpul secara mendalam melalui rekursi, sedangkan BFS menggunakan antrian (queue) untuk mengunjungi simpul secara bertingkat. Program juga dilengkapi dengan fungsi untuk mereset status kunjungan simpul dan mencetak hasil traversal dari kedua metode tersebut.

4. Unguided

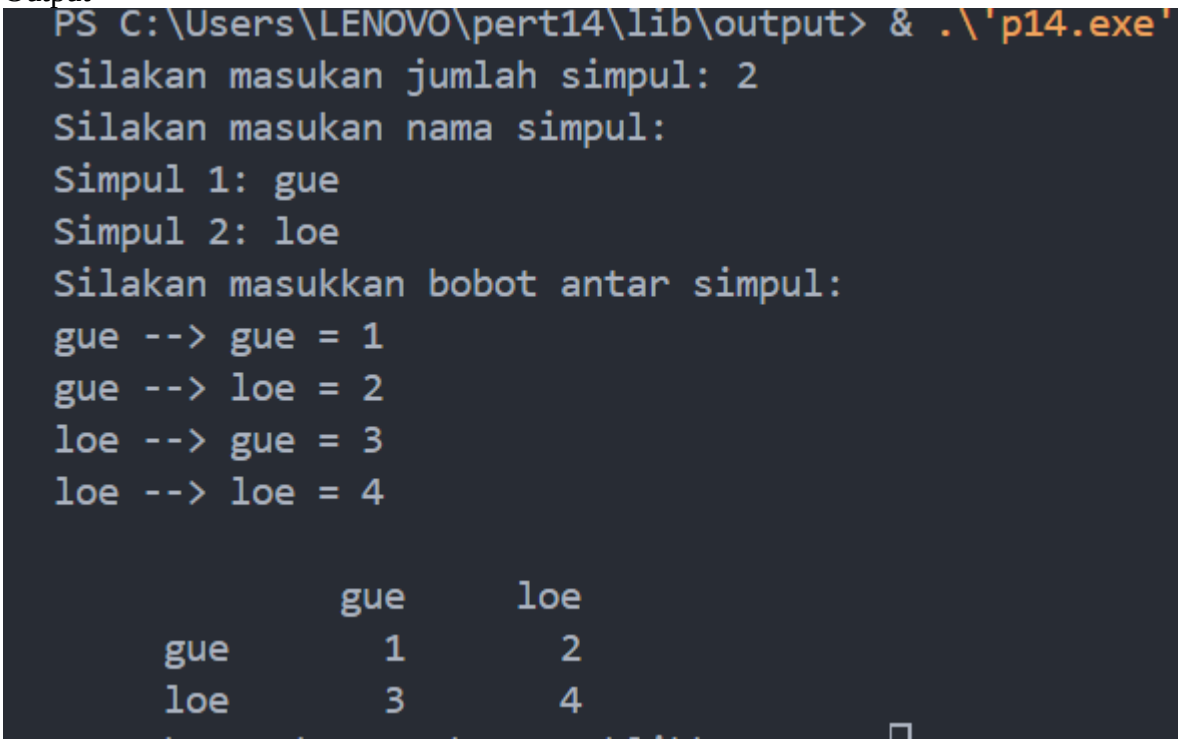
a. Unguided 1

Source Code

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

void printMatrix(const vector<vector<int>> &matrix, const vector<string> &nodes)
{
    int n = nodes.size();

    cout << setw(8) << "";
    for (int i = 0; i < n; i++)
    {
        cout << setw(8) << nodes[i];
    }
    cout << endl;

    for (int i = 0; i < n; i++)
    {
        cout << setw(8) << nodes[i];
        for (int j = 0; j < n; j++)
        {
            cout << setw(8) << matrix[i][j];
        }
        cout << endl;
    }
}

int main()
{
    int numNodes;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> numNodes;

    vector<string> nodes(numNodes);
    cout << "Silakan masukan nama simpul:\n";
    for (int i = 0; i < numNodes; i++)
    {
        cout << "Simpul " << i + 1 << ": ";
        cin >> nodes[i];
```

```
    }

    vector<vector<int>> weights(numNodes, vector<int>(numNodes, 0));

    cout << "Silakan masukkan bobot antar simpul:\n";
    for (int i = 0; i < numNodes; i++)
    {
        for (int j = 0; j < numNodes; j++)
        {
            if (i != j || (i == j && weights[i][j] == 0))
            {
                cout << nodes[i] << " --> " << nodes[j] << " = ";
                cin >> weights[i][j];
            }
        }
    }

    cout << "\n";
    printMatrix(weights, nodes);

    return 0;
}
```

Output



Deskripsi

Program C++ ini bertujuan untuk membuat dan menampilkan matriks bobot graf berbobot yang diinputkan oleh pengguna. Program dimulai dengan meminta pengguna untuk memasukkan jumlah simpul (node) dalam graf. Setelah itu, pengguna diminta untuk memberikan nama untuk setiap simpul. Kemudian, program meminta pengguna untuk mengisi bobot antar simpul-simpul tersebut, dengan setiap bobot dimasukkan dalam matriks dua dimensi yang mewakili graf tersebut. Di dalam matriks ini, setiap elemen matrix[i][j] menunjukkan bobot dari sisi yang menghubungkan simpul i ke simpul j. Matriks ini kemudian dicetak dengan format yang rapi, di mana nama-nama simpul ditampilkan di header kolom dan baris, sementara bobot-bobot sisi

antar simpul ditampilkan di sisa tabel. Program ini mengasumsikan bahwa bobot sisi untuk simpul yang sama dengan dirinya (misalnya dari simpul A ke A) diinputkan sebagai 0 (kecuali jika graf berarah dan memiliki bobot tertentu pada diagonal).

a. Unguided 2

Source Code

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

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

int main()
{
    int numNodes, numEdges;

    cout << "Masukkan jumlah simpul: ";
    cin >> numNodes;
    cout << "Masukkan jumlah sisi: ";
    cin >> numEdges;

    vector<vector<int>> adjacencyMatrix(numNodes, vector<int>(numNodes, 0));

    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < numEdges; i++)
    {
        int node1, node2;
        cin >> node1 >> node2;

          adjacencyMatrix[node1 - 1][node2 - 1] = 1;
          adjacencyMatrix[node2 - 1][node1 - 1] = 1;
      }

      cout << "\nAdjacency Matrix:\n";
      printAdjacencyMatrix(adjacencyMatrix);

```

```
107.       return 0;
108.   }
```

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
```

Deskripsi

Program C++ ini bertujuan untuk membuat dan menampilkan matriks kedekatan (adjacency matrix) dari sebuah graf tak berarah. Program dimulai dengan meminta pengguna untuk memasukkan jumlah simpul (nodes) dan jumlah sisi (edges) yang ada pada graf. Setelah itu, pengguna diminta untuk memasukkan pasangan simpul yang terhubung oleh sisi. Setiap pasangan simpul yang terhubung akan mengisi elemen-elemen tertentu pada matriks kedekatan dengan nilai 1, yang menunjukkan adanya sisi antara simpul-simpul tersebut, sementara elemen lainnya tetap bernilai 0. Matriks ini kemudian dicetak dengan format yang sesuai, di mana setiap baris dan kolom mewakili simpul, dan nilai pada elemen matriks menunjukkan apakah ada sisi yang menghubungkan simpul-simpul tersebut. Matriks ini akan simetris karena graf yang digunakan adalah graf tak berarah, artinya jika ada sisi dari simpul A ke simpul B, maka sisi dari simpul B ke simpul A juga ada.