LAPORAN PRAKTIKUM
Modul 14
“GRAPH”

Disusun Oleh:
Faishal Arif Setiawan -2311104066
Kelas -SE-07-02

Dosen :
Wahyu Andi Saputra.S.PD.,M.ENG

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. **Tujuan**

   1.Memahami konsep *graph*

   2. Mengimplementasikan *graph* dengan menggunakan *pointer*.

2. **Landasan Teori**

   Graph merupakan himpunan tidak kosong dari *node* (*vertec*) dan garis penghubung (*edge*). Contoh sederhana tentang *graph*, yaitu antara Tempat Kost Anda dengan *Common Lab*. Tempat Kost Anda dan *Common Lab* merupakan *node* (*vertec*). Jalan yang menghubungkan tempat Kost dan *Common Lab* merupakan garis penghubung antara keduanya (*edge*).

   Jenis graph:

   1.Grap berarah

   Merupakan graph dimana tiap node memiliki edge yang memiliki arah,kemana arah node tersebut di hubungkan.

   2.Graph tak berarah

   Merupakan graph dimana tiap node memiliki edge yang dihubungkan ke node lain tanpa arah.

3. **Guided**

```cpp
#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:

```
PS D:\struktur data pemograman\MODUL14\output> & .\'guidedmd14.exe'
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
PS D:\struktur data pemograman\MODUL14\output>
```

4. Unguided

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

using namespace std;

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

    vector<string> kota(n);
    vector<vector<int>> graph(n, vector<int>(n, 0));

    // Input nama-nama kota
    for (int i = 0; i < n; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> ws;
        getline(cin, kota[i]);
    }


    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i != j) {
                cout << kota[i] << " --> " << kota[j] << ": ";
                cin >> graph[i][j];
            }
        }
    }


    cout << "\nMatriks Bobot:\n";
    cout << setw(10) << " ";
    for (const auto& k : kota) {
        cout << setw(10) << k;
    }
    cout << endl;

    for (int i = 0; i < n; i++) {
        cout << setw(10) << kota[i];
        for (int j = 0; j < n; j++) {
            cout << setw(10) << graph[i][j];
        }
        cout << endl;
    }

    return 0;
}

```

Output:

```
PS D:\struktur data pemograman\MODUL14\output> & .\'unguided.exe'
Silakan masukkan jumlah simpul: 2
Simpul 1: Bali
Simpul 2: Palu
Bali --> Palu: 3
Palu --> Bali: 4

Matriks Bobot:
                Bali      Palu
      Bali         0         3
      Palu         4         0
PS D:\struktur data pemograman\MODUL14\output>
```

Telkom University Purwokerto

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

int main() {
    int jumlahSimpul, jumlahSisi;

    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan jumlah sisi: ";
    cin >> jumlahSisi;

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

    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < jumlahSisi; i++) {
        int simpul1, simpul2;
        cin >> simpul1 >> simpul2;

        adjacencyMatrix[simpul1 - 1][simpul2 - 1] = 1;
        adjacencyMatrix[simpul2 - 1][simpul1 - 1] = 1;
    }

    cout << "\nAdjacency Matrix:\n";
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << adjacencyMatrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}

```

Output:

```
PS D:\struktur data pemograman\MODUL14\output> & .\'unguided2.exe'
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:\struktur data pemograman\MODUL14\output>
```

5. **Kesimpulan**

Dalam praktikum ini, telah memahami konsep dasar dari graph, termasuk elemen-elemen penting seperti node (vertex) dan garis penghubung (edge). Graph dapat dibedakan menjadi graph berarah dan tidak berarah, yang masing-masing memiliki karakteristik tersendiri dalam menghubungkan node. Implementasi graph menggunakan pointer memberikan fleksibilitas dalam merepresentasikan struktur data yang dinamis, sehingga cocok untuk memodelkan hubungan antar data dalam berbagai aplikasi. Praktikum ini juga membantu memahami cara memvisualisasikan dan mengolah graph untuk menghasilkan representasi yang sesuai dengan kebutuhan aplikasi nyata.