LAPORAN PRAKTIKUM

MODUL 14

Graph

Disusun Oleh:

Muhammad Ikhsan Al Hakim (2311104064)

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

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

II. LANDASAN TEORI

1. 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).

III. GUIDED

1. Guided

Code:

```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:

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
PS D:\buat struktur data\pertemuan 11>
```

IV. UNGUIDED

Unguided 1;

Code:

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

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

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

    vector<vector<int>> bobot(jumlah_simpul, vector<int>(jumlah_simpul, 0));

    cout << "Silakan masukkan bobot antar simpul" << endl;
    for (int i = 0; i < jumlah_simpul; i++) {
        for (int j = 0; j < jumlah_simpul; j++) {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            cin >> bobot[i][j];
        }
    }

    cout << endl;
    cout << setw(10) << " ";
    for (const auto& s : simpul) {
        cout << setw(10) << s;
    }
    cout << endl;

    for (int i = 0; i < jumlah_simpul; i++) {
        cout << setw(10) << simpul[i];
        for (int j = 0; j < jumlah_simpul; j++) {
            cout << setw(10) << bobot[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
Silakan masukkan jumlah simpul: 2
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
PS D:\buat struktur data\pertemuan 11>
```

Unguided 2

Code:

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

using namespace std;

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

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

    cout << "Masukkan pasangan simpul yang terhubung:" << endl;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        adjacencyMatrix[u - 1][v - 1] = 1;
        adjacencyMatrix[v - 1][u - 1] = 1;
    }

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

    return 0;
}
```

Output:

```
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul yang terhubung:
1 2
1 3
2 4
4 3
Adjacency Matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
PS D:\buat struktur data\pertemuan 11>
```

KESIMPULAN:

Graf adalah struktur data yang merepresentasikan hubungan antar objek melalui himpunan simpul (vertex) dan sisi (edge). Graf digunakan untuk memodelkan berbagai masalah, seperti jaringan, hubungan sosial, dan peta. Secara formal, graf dinyatakan sebagai pasangan G-(V,E)G = (V, E), dengan VV sebagai himpunan simpul dan EE sebagai himpunan sisi.

Graf memiliki beberapa jenis berdasarkan karakteristiknya:

1. Graf Tak Berarah: Sisi tidak memiliki arah, sehingga hubungan dua simpul bersifat timbal ballk.
2. Graf Berarah: Sisi memiliki arah tertentu dari satu simpul ke simpul lain.
3. Graf Berbobot: Setiap sisi memiliki bobot yang merepresentasikan nilai tertentu, seperti jarak atau biaya.

4. Graf Lengkap: Semua simpul saling terhubung.
5. Graf Bipartit: Simpul terbagi dalam dua kelompok, dengan sisi hanya menghubungkan simpul dari kelompok yang berbeda.
Jenis-jenis ini memungkinkan graf digunakan untuk berbagai aplikasi sesuai kebutuhan