LAPORAN PRAKTIKUM

PERTEMUAN 14

14_GRAP0048

Nama :

Ilham Lii Assidaq (2311104068)

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

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

II. TOOL

Dev C++

III. DASAR TEORI

Konsep dasar graf dapat dipahami sebagai sekumpulan simpul (vertex) yang saling terhubung melalui sisi (edge) dalam struktur data graf. Dalam modul ini, Anda akan mempelajari dasar-dasar graf, termasuk berbagai jenisnya, seperti graf berarah dan tidak berarah, serta cara menggunakan pointer untuk mengelolanya. Hubungan antar simpul dapat direpresentasikan dengan teknik seperti matriks ketetanggaan dan multilist. Selain itu, terdapat algoritma yang digunakan sebagai metode penelusuran untuk memproses data dalam graf, yaitu Topological Sort, Breadth-First Search (BFS), dan Depth-First Search (DFS). Dengan pemahaman ini, modul ini akan mengajarkan penggunaan ADT graf, yang mencakup representasi struktur data, algoritma penelusuran, dan pembuatan kode untuk aplikasi praktis seperti perhitungan jalur atau pengolahan data terstruktur.

IV. GUIDED

1. GUIDED 1

```
#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;
}
```

V. UNGUIDED

1. UNGUIDED

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

using namespace std;

int main() {
    int jumlahSimpul;

    cout << "Silakan masukan jumlah simpul (minimal 1): ";
    while (true) {
        cin >> jumlahSimpul;
        if (cin.fail() || jumlahSimpul < 1) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Input tidak valid. Silakan masukkan angka positif: ";
        } else {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            break;
        }
    }

    vector<string> simpul(jumlahSimpul);

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

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

    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            while (true) {
                cin >> matriks[i][j];
                if (cin.fail()) {
                    cin.clear();
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    cout << "Input tidak valid. Silakan masukkan angka: ";
                } else {
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
                    break;
                }
            }
        }
    }

    cout << "\nMatriks Bobot:\n";
    cout << setw(8) << " ";
    for (const auto& s : simpul) {
        cout << setw(8) << s;
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << setw(8) << simpul[i];
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << setw(8) << matriks[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

KESIMPULAN

Kesimpulan yang bisa saya jelaskan dari modul 14 ini adalah bahwa modul ini membahas tentang graph, salah satu struktur data penting untuk mengatur data yang saling terhubung. Dari konsep dasar, jenis-jenis graph, hingga cara representasinya, semuanya dijelaskan dengan detail. Saya juga diajarkan cara kerja algoritma seperti Topological Sort, BFS, dan DFS untuk menjelajah graph. Modul ini mengajarkan bagaimana membuat dan mengkode graph menggunakan pendekatan dinamis seperti multilist, yang sangat cocok untuk data yang fleksibel. Setelah memahami modul ini, saya siap untuk membuat aplikasi yang membutuhkan pengolahan data dengan hubungan yang kompleks!