LAPORAN PRAKTIKUM
MODUL 14
GRAPH

Disusun Oleh:

Satria Putra Dharma Prayudha - 21104036

SE07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan

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

B. Landasan Teori

Landasan teori ini berdasarkan pada modul pembelajaran praktikum struktur data kali ini :

2.1 Graph

2.1.1 Pengertian

Graph adalah salah satu struktur data fundamental yang merepresentasikan himpunan simpul (node atau verteks) dan sisi (edge) yang menghubungkan simpul-simpul tersebut. Graph sering digunakan untuk memodelkan berbagai permasalahan nyata, seperti jaringan jalan, sistem transportasi, jaringan komputer, hingga urutan pengerjaan tugas dalam proyek. Sebagai contoh, tempat tinggal Anda dan laboratorium adalah simpul, sedangkan jalan yang menghubungkan keduanya adalah sisi.

Graph dapat digunakan untuk menggambarkan hubungan antara objek atau entitas, baik hubungan satu arah (graph berarah) maupun hubungan dua arah (graph tidak berarah). Graph juga sering dimanfaatkan dalam pengembangan algoritma yang melibatkan pencarian jalur terpendek, pengelompokan, dan analisis jaringan.

2.1.2 Jenis-Jenis Graph

A. Graph Berarah (Directed Graph)

Graph berarah adalah graph di mana setiap sisi memiliki arah tertentu. Pada graph ini, jika terdapat hubungan dari simpul A ke simpul B, maka arah hubungan tersebut spesifik dari A ke B. Contohnya, jika A → B, belum tentu ada hubungan B → A kecuali dinyatakan secara eksplisit. Dalam implementasi, graph ini sering direpresentasikan dengan panah satu arah pada sisi-sisinya.

B. Graph Tidak Berarah (Undirected Graph)

Graph tidak berarah adalah graph di mana sisi-sisi yang menghubungkan simpul tidak memiliki arah tertentu. Jika terdapat

hubungan antara simpul A dan simpul B, maka hubungan tersebut dapat dianggap dua arah (A ↔ B). Representasi ini lebih sederhana dan umum digunakan pada kasus seperti jaringan sosial atau hubungan kerjasama.

Selain arah, sisi dalam graph sering diberi bobot (weighted), misalnya untuk menunjukkan jarak atau biaya tertentu antara dua simpul.

2.1.3 Representasi Graph

Untuk mengimplementasikan graph dalam program, diperlukan representasi yang sesuai dengan kebutuhan. Berikut adalah beberapa metode representasi graph:

1. **Matrix Adjacency (Matriks Ketetanggaan)**

   Matriks ketetanggaan adalah matriks dua dimensi berukuran n×n, di mana nnn adalah jumlah simpul dalam graph. Elemen [i][j] dalam matriks menunjukkan apakah terdapat hubungan antara simpul i dan j. Matriks ini sangat cocok digunakan pada graph yang memiliki jumlah simpul kecil hingga menengah.

2. **Multilist**

   Multilist adalah struktur data dinamis yang menggunakan pointer untuk merepresentasikan simpul dan sisi. Karena sifat dinamisnya, multilist mampu menangani perubahan struktur graph dengan lebih fleksibel, seperti menambahkan atau menghapus simpul dan sisi.

2.1.4 Penelusuran Graph

Penelusuran atau traversal adalah proses mengunjungi simpul-simpul dalam graph untuk mencari informasi tertentu. Dua metode penelusuran yang umum digunakan adalah:

1. **Breadth First Search (BFS)**

   BFS adalah algoritma penelusuran yang bekerja dengan cara mengunjungi simpul secara bertahap sesuai tingkatan (level) dari simpul akar. BFS menggunakan struktur data queue untuk menyimpan

simpul yang akan dikunjungi. Metode ini cocok untuk mencari jalur terpendek dalam graph tidak berbobot.

Algoritma BFS secara umum:

a. Masukkan simpul awal ke dalam queue.
b. Selama queue tidak kosong:
    - Ambil simpul dari depan queue.
    - Tandai simpul tersebut sebagai telah dikunjungi.
    - Masukkan semua tetangganya yang belum dikunjungi ke dalam queue.

2. Depth First Search (DFS)

DFS adalah algoritma penelusuran yang bekerja dengan cara menyusuri simpul secara mendalam pada suatu cabang sebelum berpindah ke cabang lainnya. DFS biasanya diimplementasikan menggunakan struktur data stack atau melalui rekursi.

Algoritma DFS secara umum:

a. Masukkan simpul awal ke dalam stack.
b. Selama stack tidak kosong:
    - Ambil simpul dari atas stack.
    - Tandai simpul tersebut sebagai telah dikunjungi.
    - Masukkan semua tetangganya yang belum dikunjungi ke dalam stack.

2.1.5 Topological Sort

Topological Sort adalah algoritma yang digunakan untuk menyusun elemen-elemen dalam graph berarah sehingga menghasilkan urutan linier berdasarkan hubungan ketergantungan (dependency). Algoritma ini digunakan pada graph berarah tanpa siklus (DAG - Directed Acyclic Graph).

Contoh aplikasi Topological Sort:

- Penyusunan kurikulum, di mana mata pelajaran memiliki prasyarat.
- Penjadwalan proyek berdasarkan urutan pengerjaan tugas.
- Penyusunan tabel dalam basis data dengan relasi referensial.

Langkah algoritma Topological Sort:

1. Identifikasi simpul yang tidak memiliki pendahulu (predecessor).
2. Masukkan simpul tersebut ke dalam urutan linier.
3. Hapus simpul dari graph, termasuk semua sisi yang terhubung dengannya.
4. Ulangi langkah 1-3 hingga seluruh simpul telah diurutkan.

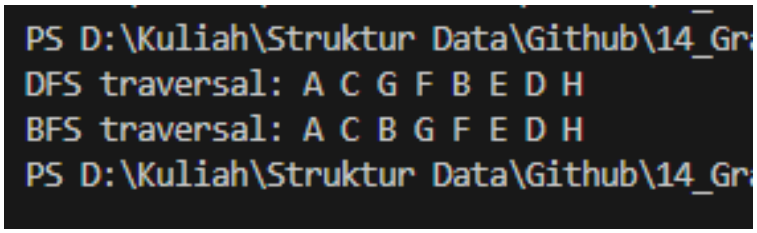
C. Guided

a. 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 :



Penjelasan : Program ini mengimplementasikan struktur data graf berarah menggunakan bahasa C++. Graf diwakili struktur Graph yang berisi simpul-simpul (ElmNode) dan setiap simpul dapat terhubung ke simpul lain melalui sisi-sisi (ElmEdge). Program ini menyediakan fungsi untuk membuat graf (CreateGraph), menambahkan simpul (InsertNode), menghubungkan simpul (ConnectNode), dan mencetak informasi graf (PrintInfoGraph). Selain itu, terdapat fungsi untuk melakukan traversal graf menggunakan metode Depth-First Search (PrintDFS) dan Breadth-First Search (PrintBFS). Pada fungsi main, graf dibangun dengan beberapa simpul dan koneksi, kemudian dilakukan traversal DFS dan BFS untuk mencetak urutan kunjungan simpul-simpul dalam graf.

D. Unguided

a. Menghitung Jarak Dari Sebuah Kota Ke Kota Lainnya

Code:

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

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

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

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

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

    cout << "\nMatriks Adjacency dengan Bobot:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
Silakan masukkan jumlah simpul: 2
Silakan masukkan nama simpul:
Simpul 1: Jakarta
Simpul 2: Bandung
Silakan masukkan bobot antar simpul:
Jakarta --> Bandung: 5
Bandung --> Jakarta: 8

Matriks Adjacency dengan Bobot:
0 5
8 0
```

Penjelasan: Program ini membaca jumlah simpul n dari sebuah graf berarah dengan bobot, kemudian menginisialisasi dan mengisi matriks adjacency menggunakan vector 2D. Pertama, pengguna diminta untuk memasukkan jumlah simpul n, kemudian nama-nama simpul disimpan dalam vektor nodes. Selanjutnya, program menginisialisasi matriks graph berukuran n x n dengan nilai awal 0. Pengguna kemudian diminta untuk memasukkan bobot antar simpul yang berbeda, yang disimpan dalam matriks graph pada posisi [i][j]. Setelah semua data dimasukkan, program mencetak matriks adjacency dengan bobot, di mana setiap elemen matriks menunjukkan bobot dari simpul i ke simpul j. vector digunakan untuk memudahkan pengelolaan dan manipulasi matriks secara dinamis, memungkinkan ukuran matriks ditentukan pada waktu runtime. Program diakhiri dengan return 0; yang menandakan bahwa eksekusi selesai dengan sukses.

b. Graf Tidak Berarah

Code:

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

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

    cout << "Masukkan jumlah sisi: ";
    cin >> e;

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

    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        adjMatrix[u - 1][v - 1] = 1;
        adjMatrix[v - 1][u - 1] = 1; // Karena graph tidak berarah
    }

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

    return 0;
}
```

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
```

Penjelasan: Program ini membaca jumlah simpul n dan jumlah sisi e dari sebuah graf tidak berarah, kemudian menginisialisasi matriks adjacency menggunakan vector 2D berukuran n x n dengan nilai awal 0. Pengguna diminta untuk memasukkan pasangan simpul yang terhubung, yang kemudian disimpan dalam matriks adjacency dengan mengatur nilai pada posisi [u-1][v-1] dan [v-1][u-1] menjadi 1, menunjukkan adanya koneksi antara simpul u dan v. Setelah semua pasangan simpul dimasukkan, program mencetak matriks adjacency yang menunjukkan koneksi antar simpul dalam graf. vector digunakan untuk memudahkan pengelolaan dan manipulasi matriks secara dinamis. Program diakhiri dengan return 0; yang menandakan bahwa eksekusi selesai dengan sukses.

E. Kesimpulan

Kesimpulan dari praktikum ini adalah bahwa implementasi graph dengan pointer memberikan fleksibilitas dalam merepresentasikan hubungan antar simpul secara dinamis. Graph, sebagai salah satu struktur data fundamental, sangat berguna untuk memodelkan berbagai permasalahan nyata seperti jaringan transportasi, sistem komputer, dan urutan pengerjaan tugas. Dalam praktikum ini, konsep graph berarah, graph tidak berarah, serta algoritma penelusuran seperti Breadth First Search (BFS) dan Depth First Search (DFS) berhasil dipahami dan diimplementasikan. Selain itu, representasi graph menggunakan matriks adjacency dan multilist memberikan kelebihan masing-masing, di mana matriks adjacency cocok untuk graph sederhana, sedangkan multilist lebih fleksibel untuk struktur yang berubah.

Selain itu, algoritma seperti Topological Sort memberikan solusi efektif untuk permasalahan ketergantungan, seperti penjadwalan dan penyusunan kurikulum. Praktikum ini juga menekankan pentingnya pemahaman implementasi graph dalam berbagai konteks aplikasi, termasuk penghitungan jarak antar simpul dan representasi graf tidak berarah. Dengan menguasai konsep dan implementasi ini, praktikan dapat memanfaatkan graph sebagai alat untuk menyelesaikan berbagai masalah kompleks dalam pengelolaan data dan analisis jaringan.