LAPORAN PRAKTIKUM

MODUL 14

“ GRAPH ”

Disusun Oleh:

Rengganis Tantri Pramudita - 2311104065

S1SE0702

Dosen :

Wahyu Andy Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

- Memahami konsep graph
- Mengimplementasikan graph dengan menggunakan pointer.

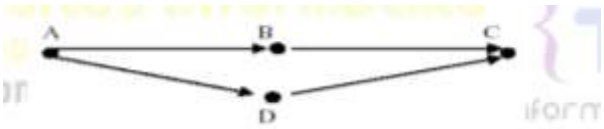
2. Landasan Teori

Graph merupakan himpunan tidak kosong dari node (vertec) dan garis penghubung (edge).

Jenis - jenis Graph:

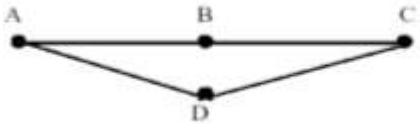
- Graph Berarah

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



- Graph Tidak Berarah

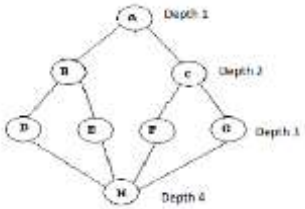
  Merupakan graph dimana tiap node memiliki edge yang dihubungkan ke node lain tanpa arah.



Metode penelusuran graph

- Breadth First Search (BFS)

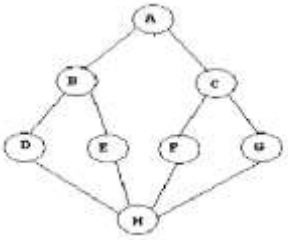
  Cara kerja algoritma ini adalah dengan mengunjungi root (depth 0) kemudian ke depth 1, 2, dan seterusnya. Kunjungan pada masing-masing level dimulai dari kiri ke kanan.



  Urutannya hasil penelusuran BFS : A B C D E F G H

- Depth First Search (DFS)

  Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian rekursif ke subtree node tersebut.



Urutannya : A B D H E F C G

3. Guided

```
#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;
};

// Fungsi untuk membuat graf baru
void CreateGraph(Graph &G) {
   G.first = NULL; // Inisialisasi graf kosong
}

// Fungsi untuk menambahkan node baru ke graf
void InsertNode(Graph &G, char X) {
   ElmNode *newNode = new ElmNode; // Alokasi memori untuk node baru
   newNode->info = X;            // Menyimpan informasi
   newNode->visited = false;     // Status awal belum dikunjungi
   newNode->firstEdge = NULL;    // Belum ada tepi
```

```
    newNode->Next = NULL;          // Node berikutnya adalah NULL

    if (G.first == NULL) {
        G.first = newNode; // Jika graf kosong, tambahkan sebagai simpul pertama
    } else {
        ElmNode *temp = G.first;
        while (temp->Next != NULL) {
            temp = temp->Next; // Cari node terakhir dalam daftar
        }
        temp->Next = newNode; // Tambahkan node baru di akhir
    }
}

// Fungsi untuk membuat hubungan antar node (menambahkan tepi)
void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge; // Alokasi memori untuk tepi baru
    newEdge->Node = N2;            // Tepi menghubungkan ke simpul N2
    newEdge->Next = N1->firstEdge; // Sisipkan di awal daftar tepi
    N1->firstEdge = newEdge;       // Perbarui daftar tepi N1
}

// Fungsi untuk mencetak informasi node dalam graf
void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " "; // Cetak informasi setiap simpul
        temp = temp->Next;         // Lanjut ke simpul berikutnya
    }
    cout << endl;
}

// Fungsi untuk mengatur ulang status visited semua simpul
void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false; // Reset visited ke false
        temp = temp->Next;
    }
}

// Fungsi untuk traversal graf menggunakan DFS
void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
```

```
        return; // Basis rekursi, jika simpul NULL, selesai
    }
    N->visited = true; // Tandai simpul telah dikunjungi
    cout << N->info << " "; // Cetak informasi simpul

    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node); // Rekursi ke simpul yang terhubung
        }
        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;
}
```

Keterangan

Kode di atas adalah implementasi graf dalam C++ yang memungkinkan traversal menggunakan DFS (Depth First Search) dan BFS (Breadth First Search). Graf direpresentasikan dengan node (simpul) dan edge (tepi), di mana setiap node bisa terhubung ke node lain. Program dimulai dengan membuat graf kosong, lalu menambahkan simpul-simpul seperti 'A', 'B', hingga 'H'. Selanjutnya, hubungan antar simpul (tepi) dibentuk, misalnya, 'A' terhubung ke 'B' dan 'C'. Traversal DFS dilakukan secara rekursif dengan

menelusuri simpul secara mendalam ke cabang sebelum berpindah ke cabang lain. Sedangkan traversal BFS menggunakan antrian untuk menjelajahi simpul secara melintang atau level per level. Hasil dari kedua traversal dicetak ke layar, menunjukkan urutan kunjungan simpul.

Outputnya:

4. Unguided

Unguided 1

```
#include <iostream>
#include <string>
#include <iomanip>
#define MAX 100
using namespace std;

int main() {
    int jumlahSimpul;
    string namaSimpul[MAX];
    int bobot[MAX][MAX];

    // Input jumlah simpul
    cout << "Silakan masukan jumlah simpul : ";
    cin >> jumlahSimpul;

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

    // Inisialisasi matriks bobot dengan 0
    for(int i = 0; i < jumlahSimpul; i++) {
        for(int j = 0; j < jumlahSimpul; j++) {
            bobot[i][j] = 0;
        }
    }

    // Input bobot antar simpul
    cout << "Silakan masukkan bobot antar simpul\n";

    // Input untuk BALI ke BALI
    cout << namaSimpul[0] << "--> " << namaSimpul[0] << " = ";
    cin >> bobot[0][0];

    // Input untuk BALI ke PALU
```

```
    cout << namaSimpul[0] << "--> " << namaSimpul[1] << " = ";
    cin >> bobot[0][1];

    // Input untuk PALU ke BALI
    cout << namaSimpul[1] << "--> " << namaSimpul[0] << " = ";
    cin >> bobot[1][0];

    // Input untuk PALU ke PALU
    cout << namaSimpul[1] << "--> " << namaSimpul[1] << " = ";
    cin >> bobot[1][1];

    // Menampilkan matriks bobot
    cout << "\n";
    cout << setw(10) << " ";
    for(int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << namaSimpul[i];
    }
    cout << "\n";

    for(int i = 0; i < jumlahSimpul; i++) {
        cout << setw(10) << namaSimpul[i];
        for(int j = 0; j < jumlahSimpul; j++) {
            cout << setw(10) << bobot[i][j];
        }
        cout << "\n";
    }

    return 0;
}
```

Keterangan:

Program di atas dibuat untuk menghitung dan menampilkan jarak antar simpul (node) dalam bentuk matriks bobot. Pertama, pengguna diminta memasukkan jumlah simpul (kota) yang akan digunakan, diikuti dengan nama-nama simpul tersebut. Kemudian, program meminta pengguna untuk mengisi bobot (jarak) antara setiap pasangan simpul, termasuk jarak dari simpul ke dirinya sendiri. Data jarak ini disimpan dalam sebuah matriks dua dimensi. Setelah semua data dimasukkan, program mencetak matriks bobot dengan nama simpul di baris dan kolom, sehingga pengguna dapat melihat hubungan dan jarak antar simpul dengan jelas. Program ini cocok untuk merepresentasikan jaringan kota atau hubungan antar node secara sederhana.

Outputnya

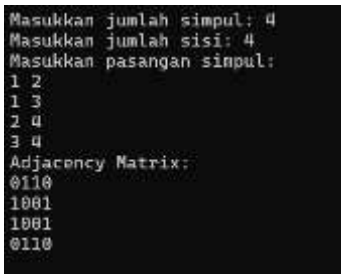
Unguided 2

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

using namespace std;

int main() {
    int n, m;

    // Input jumlah simpul dan sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> n;
    cout << "Masukkan jumlah sisi: ";
    cin >> m;

    // Inisialisasi adjacency matrix dengan nilai 0
    vector<vector<int>> adj(n, vector<int>(n, 0));

    // Input pasangan simpul yang terhubung
    cout << "Masukkan pasangan simpul:\n";
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        // Kurangi 1 karena indeks array dimulai dari 0
        u--;
        v--;
        adj[u][v] = 1;
        adj[v][u] = 1; // Karena graf tidak berarah
    }

    // Tampilkan adjacency matrix
    cout << "Adjacency Matrix:\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << adj[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Keterangan
Cara kerja kode ini secara garis besar adalah sebagai berikut:

1. Input data: Program meminta pengguna untuk memasukkan jumlah simpul dan sisi dalam graf, serta pasangan simpul yang terhubung oleh sisi.
2. Inisialisasi matriks: Program membuat sebuah matriks kosong dengan ukuran sesuai jumlah simpul. Setiap elemen dalam matriks diinisialisasi dengan nilai 0, yang menandakan tidak ada sisi di antara dua simpul.
3. Pembentukan matriks adjacency: Program kemudian mengisi matriks adjacency berdasarkan input pasangan simpul. Jika ada sisi yang menghubungkan simpul i dan j, maka elemen pada baris ke-i kolom ke-j dan baris ke-j kolom ke-i dalam matriks akan diisi dengan nilai 1.
4. Output: Program mencetak matriks adjacency ke layar, sehingga kita dapat melihat representasi visual dari graf tersebut.

Outputnya



5. Kesimpulan

Graf adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek-objek. Objek-objek ini disebut simpul (node) dan hubungan antar simpul disebut sisi (edge). Graf dapat digunakan untuk memodelkan berbagai macam hal, mulai dari jaringan sosial, peta jalan, hingga algoritma pencarian. Ada beberapa cara untuk merepresentasikan graf, salah satunya adalah menggunakan matriks adjacency. Matriks adjacency adalah sebuah matriks yang menunjukkan keberadaan sisi antara setiap pasangan simpul. Graf memiliki banyak aplikasi dalam ilmu komputer, seperti dalam pengembangan algoritma pencarian jalur terpendek, analisis jaringan sosial, dan pemodelan sistem kompleks.