Telkom University Purwokerto

Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHubyang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. AsistenPraktikum:4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folderdalam repositoryakan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwalpraktikumadalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktuini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem- solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

LAPORAN PRAKTIKUM
MODUL 14
GRAPH

Disusun Oleh :

Zaenarif Putra ‘Ainurdin – 2311104049

Kelas :

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng

PROGRAM STUDI SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

I. TUJUAN

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

II. LANDASAN 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.

III. GUIDE

1. Guide

   a. Code Program

```
#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;
}
```

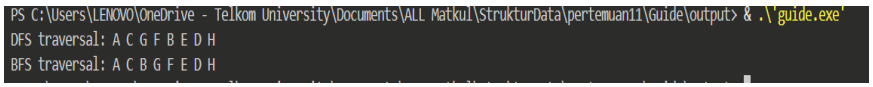
b. Penjelasan Code Program

- Struktur ElmNode: Struktur ini menggambarkan sebuah simpul dalam graf. Setiap simpul menyimpan informasi karakter (info), status kunjungan (visited), pointer ke tepi pertama (firstEdge), dan pointer ke simpul selanjutnya (Next).
- Struktur ElmEdge: Struktur ini menggambarkan tepi yang menghubungkan dua node. Setiap tepi memiliki penunjuk ke node yang terhubung dan penunjuk ke tepi selanjutnya.
- Struktur Graph: Struktur ini menggambarkan graf itu sendiri, dengan sebuah pointer yang menunjuk ke simpul pertama (first).
- void BuatGraf(Graph &G): Fungsi ini bertugas untuk menginisialisasi graf dengan mengatur pointer first menjadi NULL, yang menunjukkan bahwa graf tersebut masih kosong.
- void InsertNode(Graph &G, char X): Fungsi ini berfungsi untuk menambahkan simpul baru ke dalam graf. Apabila graf masih kosong, simpul baru akan menjadi simpul yang pertama. Jika sudah ada simpul lain, maka simpul baru akan ditambahkan di akhir daftar simpul yang ada.
- void ConnectNode(ElmNode *N1, ElmNode *N2): Fungsi ini menghubungkan dua simpul dengan menambahkan tepi dari N1 ke N2. Tepi baru ditambahkan di awal daftar tepi dari N1.
- void PrintInfoGraph(Graph G): Fungsi ini mencetak informasi dari setiap simpul dalam graf.
- void ResetVisited(Graph &G): Fungsi ini mengatur ulang status

kunjungan semua simpul dalam graf menjadi false.

- void PrintDFS(Graph G, ElmNode *N): Fungsi ini melakukan traversal graf menggunakan metode Depth-First Search (DFS). Fungsi ini menandai simpul yang dikunjungi dan mencetak informasi simpul tersebut.
- void PrintBFS(Graph G, ElmNode *N): Fungsi ini melakukan traversal graf menggunakan metode Breadth-First Search (BFS). Fungsi ini menggunakan antrian untuk mengelola simpul yang akan dikunjungi dan menandai simpul yang telah dikunjungi.
- int main(): Fungsi utama yang menjalankan program. Di sini, graf dibuat, simpul ditambahkan, dan simpul dihubungkan. Kemudian, traversal DFS dan BFS dilakukan dan hasilnya dicetak.

c. Output



IV. UNGUIDED

1. Unguided 1

a. Code Program :

```
#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;
}
```

b. Penjelasan Code Program :

- #include <iostream> digunakan untuk input dan output, #include <vector> untuk menggunakan struktur data vektor, #include <iomanip> untuk manipulasi format output, dan #include <limits> untuk mengatur batasan input.
- Variabel jumlahSimpul dideklarasikan untuk menyimpan jumlah simpul

yang akan dimasukkan oleh pengguna.

- Program meminta pengguna untuk memasukkan jumlah simpul. Menggunakan loop while untuk memastikan input valid, yaitu angka positif. Jika input tidak valid, program membersihkan status kesalahan dengan cin.clear() dan mengabaikan sisa input dengan cin.ignore(), kemudian meminta pengguna untuk memasukkan angka positif lagi.
- Vektor simpul diinisialisasi dengan ukuran jumlahSimpul untuk menyimpan nama-nama simpul yang akan dimasukkan oleh pengguna.
- Menggunakan loop for untuk meminta pengguna memasukkan nama setiap simpul. Nama simpul disimpan dalam vektor simpul pada indeks yang sesuai.
- Matriks bobot (adjacency matrix) diinisialisasi dengan ukuran jumlahSimpul x jumlahSimpul, diisi dengan nilai 0. Ini menunjukkan bahwa tidak ada bobot yang terhubung antar simpul pada awalnya.
- Menggunakan nested loop untuk meminta pengguna memasukkan bobot antar setiap pasangan simpul. Program menampilkan pesan yang menunjukkan hubungan antara simpul yang sedang dimasukkan bobotnya. Jika input tidak valid, program membersihkan status kesalahan dan meminta pengguna untuk memasukkan angka yang valid.
- Program menampilkan header untuk matriks bobot dengan menggunakan setw untuk mengatur lebar kolom agar output lebih rapi. Kemudian, menggunakan loop untuk menampilkan isi matriks bobot berdasarkan input yang diberikan oleh pengguna, di mana setiap baris mewakili bobot dari simpul tertentu ke semua simpul lainnya.
- Kode ini menandakan akhir dari fungsi main dan mengembalikan nilai 0, yang menunjukkan bahwa program telah selesai dijalankan dengan sukses tanpa kesalahan.

c. Output :

```
Silakan masukan jumlah simpul (minimal 1): 2
Simpul 1: Karawang
Simpul 2: Medan
Karawang --> Karawang = 0
Karawang --> Medan = 200
Medan --> Karawang = 300
Medan --> Medan = 0

Matriks Bobot:
        Karawang   Medan
Karawang       0     200
   Medan     300       0
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan11\Unguided\output>
```

2. Unguided 2

a. Code Program :

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <sstream>

using namespace std;
```

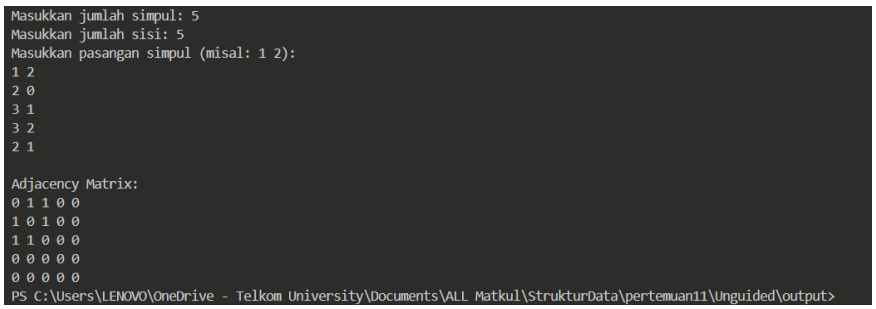
```
int main() {
    int jumlahSimpul, jumlahSisi;

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

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

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

    cout << "Masukkan pasangan simpul (misal: 1 2):\n";
    cin.ignore();
    for (int i = 0; i < jumlahSisi; i++) {
        string input;
        getline(cin, input);

        istringstream iss(input);
        int u, v;
        iss >> u >> v;

        u--;
        v--;

        matriks[u][v] = 1;
        matriks[v][u] = 1;
    }

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

    return 0;
}
```

b. Penjelasan Code Program

- Kode ini mengimpor pustaka yang diperlukan untuk program. #include <iostream> digunakan untuk input dan output, #include <vector> untuk menggunakan struktur data vektor, #include <iomanip> untuk manipulasi format output, dan #include <sstream> untuk memproses string sebagai aliran data.
- Dua variabel jumlahSimpul dan jumlahSisi dideklarasikan untuk menyimpan jumlah simpul dan jumlah sisi yang akan dimasukkan oleh pengguna.

- Program meminta user untuk memasukkan jumlah simpul yang akan digunakan dalam graf. Nilai ini disimpan dalam variabel jumlahSimpul.
- Program meminta user untuk memasukkan jumlah sisi yang akan menghubungkan simpul-simpul dalam graf. Nilai ini disimpan dalam variabel jumlahSisi.
- Matriks bobot (adjacency matrix) diinisialisasi dengan ukuran jumlahSimpul x jumlahSimpul, diisi dengan nilai 0. Ini menunjukkan bahwa tidak ada hubungan antar simpul pada awalnya.
- Program meminta pengguna untuk memasukkan pasangan simpul yang terhubung. cin.ignore() digunakan untuk mengabaikan newline yang tersisa di buffer setelah input jumlah sisi. Loop for digunakan untuk mengulangi proses input sebanyak jumlah sisi yang ditentukan.
- istringstream digunakan untuk memproses string input yang berisi pasangan simpul. Dua variabel u dan v dideklarasikan untuk menyimpan simpul yang terhubung. Nilai dari input dibaca ke dalam u dan v.
- Indeks u dan v dikurangi 1 untuk menyesuaikan dengan indeks matriks yang dimulai dari 0, karena pengguna biasanya memasukkan simpul mulai dari 1.
- Matriks diisi dengan 1 pada posisi yang sesuai untuk menunjukkan bahwa ada hubungan antara simpul u dan v. Karena graf ini tidak berarah, hubungan dicatat di kedua arah.
- Program menampilkan matriks adjacency yang telah diisi. Menggunakan nested loop untuk mencetak setiap elemen dari matriks, di mana setiap baris mewakili simpul dan setiap kolom menunjukkan hubungan dengan simpul lainnya.
- Kode ini menandakan akhir dari fungsi main dan mengembalikan nilai 0, yang menunjukkan bahwa program telah selesai dijalankan dengan sukses tanpa kesalahan.

c. Output



V. KESIMPULAN

Kesimpulan yang bisa dijelaskan modul 14 ini ngebahas tentang graph, salah satu struktur data penting buat ngatur data yang saling terhubung. Dari konsep dasar, jenis-jenis graph, sampai cara representasinya, semua dijelasin dengan detail. Kita juga diajarin cara kerja algoritma kayak Topological Sort, BFS, dan DFS buat menjelajah graph. Modul ini ngajarin gimana bikin dan ngoding graph pakai pendekatan dinamis seperti multilist, yang cocok banget buat data yang fleksibel. Jadi, setelah paham modul ini, kita siap deh buat bikin aplikasi yang butuh ngolah data dengan hubungan kompleks!