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 :

Izzaty Zahara Br Barus – 2311104052

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

Graph adalah struktur data yang terdiri dari himpunan node (sering disebut verteks) dan garis penghubung (edge) yang merepresentasikan hubungan antara node. Graph dapat digunakan untuk memodelkan berbagai hubungan dalam kehidupan nyata, seperti jaringan transportasi, hubungan dalam media sosial, atau ketergantungan dalam sebuah proyek. Berdasarkan arah hubungannya, graph terbagi menjadi graph berarah (directed graph), di mana edge memiliki arah tertentu, dan graph tidak berarah (undirected graph), di mana edge tidak memiliki arah.

Representasi graph dapat dilakukan melalui matriks ketetanggaan (adjacency matrix) atau daftar ketetanggaan (adjacency list). Untuk kebutuhan yang lebih dinamis, graph sering direpresentasikan menggunakan multilist. Metode pencarian dalam graph meliputi Breadth First Search (BFS), yang menelusuri graph berdasarkan level atau kedalaman tertentu, dan Depth First Search (DFS), yang menelusuri graph secara rekursif ke cabang terdalam terlebih dahulu. Graph juga digunakan dalam algoritma topological sorting untuk menyusun elemen secara linier berdasarkan keterurutan parsial, sering kali diterapkan dalam manajemen proyek atau struktur kurikulum.

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');
```

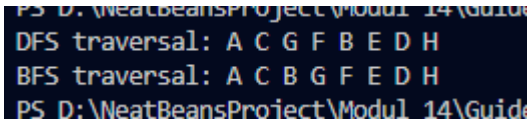
```
    ElmNode *A = G.first;
    ElmNode *B = A->Next;
    ElmNode *C = B->Next;
    ElmNode *D = C->Next;
    ElmNode *E = D->Next;
    ElmNode *F = E->Next;
    ElmNode *G1 = F->Next;
    ElmNode *H = G1->Next;

    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

- Program ini mengimplementasikan graf menggunakan struktur data simpul (node) dan sisi (edge). Graf direpresentasikan sebagai kumpulan simpul yang saling terhubung, di mana setiap simpul memiliki atribut seperti informasi (info), penanda kunjungan (visited), daftar sisi pertama (firstEdge), dan pointer ke simpul berikutnya. Program menyediakan fungsi untuk membuat graf kosong (*CreateGraph*), menambahkan simpul baru (*InsertNode*), menghubungkan dua simpul (*ConnectNode*), mengatur ulang status kunjungan simpul (*ResetVisited*), serta melakukan traversal graf menggunakan metode *depth-first search* (DFS) dan *breadth-first search* (BFS). Dalam fungsi utama, graf dibuat dengan simpul-simpul bernama 'A' hingga 'H' dan dihubungkan sesuai dengan struktur yang telah ditentukan. Traversal DFS dilakukan dengan menjelajahi simpul secara mendalam hingga cabang terakhir sebelum pindah ke cabang lain, sementara traversal BFS menjelajahi simpul secara berurutan berdasarkan levelnya. Hasil traversal ditampilkan di layar untuk menunjukkan urutan kunjungan simpul oleh masing-masing

metode.

c. Output



IV. UNGUIDED

1. Unguided 1

a. Code Program :

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

using namespace std;

int main() {
   int numCities;

   cout << "Silakan masukkan jumlah simpul: ";
   cin >> numCities;

   vector<string> cityNames(numCities);
            vector<vector<int>>      adjacencyMatrix(numCities,
vector<int>(numCities, 0));

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

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

   cout << endl << "Adjacency Matrix:" << endl;
   cout << setw(8) << " ";
   for (const auto &name : cityNames) {
      cout << setw(8) << name;
   }
```

```
        cout << endl;

        for (int i = 0; i < numCities; ++i) {
            cout << setw(8) << cityNames[i];
            for (int j = 0; j < numCities; ++j) {
                cout << setw(8) << adjacencyMatrix[i][j];
            }
            cout << endl;
        }

        return 0;
    }
```

b. Penjelasan Code Program :

- Program ini adalah implementasi graf berbasis adjacency matrix dalam bahasa C++ untuk menghitung bobot atau jarak antar simpul (node) berdasarkan input pengguna. Program dimulai dengan meminta jumlah simpul (kota) yang akan dimasukkan. Selanjutnya, pengguna diminta untuk memasukkan nama masing-masing simpul. Setelah itu, program meminta pengguna untuk memasukkan bobot antar simpul, yang merepresentasikan jarak dari satu kota ke kota lain. Input bobot ini diolah dalam bentuk matriks dua dimensi yang disebut adjacency matrix.
- Adjacency matrix ini adalah representasi matriks berbentuk persegi, di mana baris dan kolomnya menunjukkan hubungan antar simpul. Nilai pada matriks di posisi baris iii dan kolom jjj menunjukkan bobot dari simpul iii ke jjj. Jika nilai bobot adalah 0, maka simpul tersebut tidak memiliki hubungan atau jarak dengan dirinya sendiri. Setelah semua input dimasukkan, program mencetak adjacency matrix, lengkap dengan nama simpul untuk memudahkan pembacaan.
- Sebagai contoh, jika pengguna memasukkan dua simpul dengan nama "BALI" dan "PALU" serta bobot masing-masing, seperti dari "BALI ke PALU" adalah 3 dan dari "PALU ke BALI" adalah 4, maka output program akan menampilkan matriks yang merepresentasikan hubungan ini. Output ini dirancang agar pengguna dapat memahami hubungan antar simpul dan jarak masing-masing dengan mudah.
-

c. 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

Adjacency Matrix:
           BALI    PALU
    BALI      0       3
    PALU      4       0
PS D:\NeatBeansProject\Modul 14\Unguided\output>
```

2. Unguided 2

   a. Code Program :

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

using namespace std;

void printAdjacencyMatrix(const vector<vector<int>>& matrix)
{
   cout << "Adjacency Matrix:" << endl;
   for (const auto& row : matrix) {
      for (const auto& val : row) {
         cout << val << " ";
      }
      cout << endl;
   }
}

int main() {
   int numNodes, numEdges;

   cout << "Masukkan jumlah simpul: ";
   cin >> numNodes;
   cout << "Masukkan jumlah sisi: ";
   cin >> numEdges;


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

   cout << "Masukkan pasangan simpul yang terhubung:" << endl;

   for (int i = 0; i < numEdges; ++i) {
```

```
            int node1, node2;
            cin >> node1 >> node2;


            node1--;
            node2--;

            adjacencyMatrix[node1][node2] = 1;
            adjacencyMatrix[node2][node1] = 1;
        }

        printAdjacencyMatrix(adjacencyMatrix);

        return 0;
    }
```

b. Penjelasan Code Program

- Program di atas merepresentasikan graf tidak berarah menggunakan adjacency matrix. Program ini menerima input jumlah simpul (nodes) dan jumlah sisi (edges), lalu pengguna diminta untuk memasukkan pasangan simpul yang saling terhubung. Matriks ketetanggaan, yang merupakan matriks n×nn \times nn×n (dengan nnn adalah jumlah simpul), diinisialisasi dengan nilai 0. Setiap elemen matriks M[i][j]M[i][j]M[i][j] akan diubah menjadi 1 jika terdapat hubungan (edge) antara simpul iii dan jjj. Karena graf ini tidak berarah, M[i][j]=M[j][i]M[i][j] = M[j][i]M[i][j]=M[j][i].
- Setelah semua pasangan simpul diinput, program mencetak adjacency matrix untuk memvisualisasikan keterhubungan antar simpul. Dalam adjacency matrix, baris dan kolom merepresentasikan simpul, dan nilai 1 menunjukkan adanya hubungan antara simpul-simpul tersebut, sedangkan nilai 0 menunjukkan tidak adanya hubungan. Program ini membantu memahami hubungan antar simpul dalam graf dengan cara yang sederhana dan sistematis.

c. Output

```
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul yang terhubung: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
```

V. KESIMPULAN

Implementasi graf dalam pemrograman dapat dilakukan menggunakan struktur data pointer dan adjacency matrix. Pendekatan dengan pointer memanfaatkan hubungan dinamis antar node melalui edge, cocok untuk graf dengan konektivitas yang fleksibel. Sementara itu, adjacency matrix adalah cara representasi yang sederhana untuk menggambarkan bobot atau hubungan antar simpul dalam bentuk tabel, meski kurang efisien untuk graf besar dengan sedikit koneksi. Metode traversal seperti DFS dan BFS memungkinkan eksplorasi graf dengan pendekatan yang berbeda, berguna untuk berbagai kasus seperti pencarian rute atau analisis hubungan. Melalui pemahaman ini, graf dapat diterapkan untuk memodelkan masalah dunia nyata secara efektif.