LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL 12 (14)

GRAPH

Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Graph didefinisikan sebagai himpunan tidak kosong dari node (atau vertec) dan garis penghubung yang disebut edge. Terdapat dua jenis graph: Graph Berarah (Directed Graph) yang memiliki edge dengan arah spesifik ke mana node dihubungkan, dan Graph Tidak Berarah (Undirected Graph) di mana edge dihubungkan tanpa arah. Dalam graph tak-berarah, jika dua node dihubungkan langsung oleh sebuah edge, keduanya dikatakan bertetangga. Representasi graph dapat dilakukan dengan Matrik Ketetanggaan (Adjacency Matrices) yang diimplementasikan sebagai Array 2 Dimensi atau Multi Linked List. Konsep penting lainnya adalah Topological Sort, yaitu proses untuk mendapatkan keterurutan linear dari elemen-elemen yang memiliki urutan partial. Prosesnya melibatkan pemilihan dan penghapusan item yang tidak memiliki predecessor (jumlah predecessor nol) secara berulang hingga himpunan elemen menjadi kosong. Terakhir, terdapat dua metode utama penelusuran graph: Breadth First Search (BFS) yang bekerja dengan mengunjungi node secara level demi level (menggunakan Queue) dan Depth First Search (DFS) yang bekerja secara rekursif ke subtree node (menggunakan Stack)

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

(“graph.h”)

```
#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED

#include <iostream>
using namespace std;

typedef char infoGraph;

struct ElmNode;
struct ElmEdge;

typedef ElmNode* adrNode;
typedef ElmEdge* adrEdge;

struct ElmNode {
   infoGraph info;
   int visited;
   adrEdge firstEdge;
```

```
    adrNode next;
};

struct ElmEdge {
    adrNode node;
    adrEdge next;
};

struct Graph {
    adrNode first;
};

// Primitif Graph
void CreateGraph(Graph &G);
adrNode AllocateNode(infoGraph X);
adrEdge AllocateEdge(adrNode N);

void InsertNode(Graph &G, infoGraph X);
adrNode FindNode(Graph G, infoGraph X);

void ConnectNode(Graph &G, infoGraph A, infoGraph B);

void PrintInfoGraph(Graph G);

// Traversal
void ResetVisited(Graph &G);
void PrintDFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

(“graph.cpp”)

```
#include "graf.h"
#include <queue>
#include <stack>

void CreateGraph(Graph &G) {
    G.first = NULL;
}

adrNode AllocateNode(infoGraph X) {
    adrNode P = new ElmNode;
    P->info = X;
```

```
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}

adrEdge AllocateEdge(adrNode N) {
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}

void InsertNode(Graph &G, infoGraph X) {
    adrNode P = AllocateNode(X);
    P->next = G.first;
    G.first = P;
}

adrNode FindNode(Graph G, infoGraph X) {
    adrNode P = G.first;
    while (P != NULL) {
        if (P->info ==X)
            return P;
        P = P->next;
    }
    return NULL;
}

void ConnectNode(Graph &G, infoGraph A, infoGraph B) {
    adrNode N1 = FindNode(G, A);
    adrNode N2 = FindNode(G, B);

    if (N1 == NULL || N2 == NULL) {
        cout << "Node tidak ditemukan!\n";
        return;
    }

    // Buat edge dari N1 ke N2
    adrEdge E1 = AllocateEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;
```

```
    // Karena undirected -> buat edge balik
    adrEdge E2 = AllocateEdge(N1);
    E2->next = N2->firstEdge;
    N2->firstEdge = E2;
}

void PrintInfoGraph(Graph G) {
    adrNode P = G.first;
    while (P != NULL) {
        cout << P->info << " -> ";
        adrEdge E = P->firstEdge;
        while (E != NULL) {
            cout << E->node->info << " ";
            E = E->next;
        }
        cout << endl;
        P = P->next;
    }
}

void ResetVisited(Graph &G) {
    adrNode P = G.first;
    while (P != NULL) {
        P->visited = 0;
        P = P->next;
    }
}

void PrintDFS(Graph &G, adrNode N) {
    if (N == NULL)
        return;

    N->visited = 1;
    cout << N->info << " ";

    adrEdge E = N->firstEdge;
    while (E != NULL) {
        if (E->node->visited == 0) {
            PrintDFS(G, E->node);
        }
        E = E->next;
    }
}
```

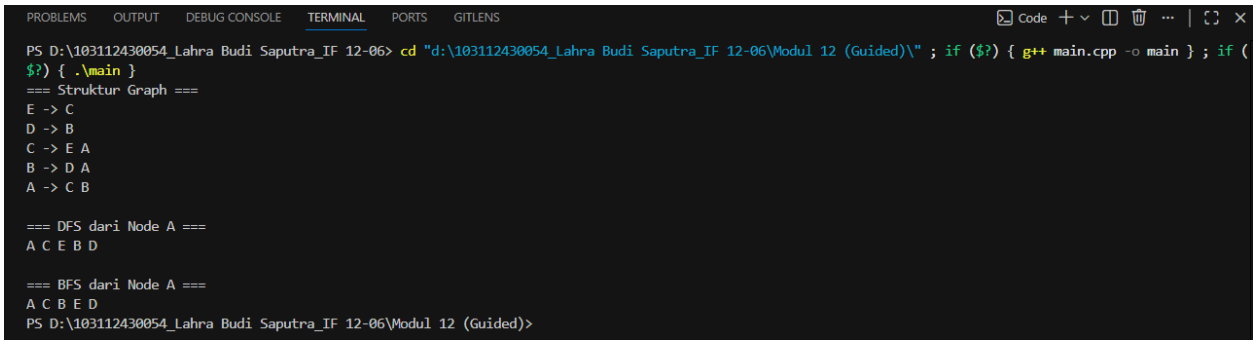
```
void PrintBFS(Graph &G, adrNode N) {
   if (N == NULL)
      return;

   queue<adrNode> Q;
   Q.push(N);

   while (!Q.empty()) {
      adrNode curr = Q.front();
      Q.pop();

      if (curr->visited == 0) {
         curr->visited = 1;
         cout << curr->info << " ";

         adrEdge E = curr->firstEdge;
         while (E != NULL) {
            if (E->node->visited == 0) {
               Q.push(E->node);
            }
            E = E->next;
         }
      }
   }
}
```

(“main.cpp”)

```
#include "graf.h"
#include "graf.cpp"
#include <iostream>
using namespace std;

int main() {
   Graph G;
   CreateGraph(G);

   // Tambah node
   InsertNode(G, 'A');
   InsertNode(G, 'B');
   InsertNode(G, 'C');
   InsertNode(G, 'D');
```

```
    InsertNode(G, 'E');

    // Hubungkan node (Graph tidak berarah)
    ConnectNode(G, 'A', 'B');
    ConnectNode(G, 'A', 'C');
    ConnectNode(G, 'B', 'D');
    ConnectNode(G, 'C', 'E');

    cout << "=== Struktur Graph ===\n";
    PrintInfoGraph(G);

    cout << "\n=== DFS dari Node A ===\n";
    ResetVisited(G);
    PrintDFS(G, FindNode(G, 'A'));

    cout << "\n\n=== BFS dari Node A ===\n";
    ResetVisited(G);
    PrintBFS(G, FindNode(G, 'A'));

    cout << endl;

    return 0;
}
```

Screenshots Output



Deskripsi:

Program di atas terdiri dari tiga file utama yang saling terintegrasi untuk mengimplementasikan graph tidak berarah menggunakan adjacency list. File **graph.h** berisi definisi struktur data graph (node dan edge) serta deklarasi fungsi-fungsi dasar dan traversal. File **graph.cpp** mengimplementasikan seluruh fungsi tersebut, termasuk pembuatan graph, penambahan dan penghubungan node, serta algoritma traversal **DFS** dan **BFS**. Sementara itu, file **main.cpp** berperan sebagai program utama yang

membangun graph, menghubungkan node-node, menampilkan struktur graph, dan menjalankan traversal untuk menunjukkan cara kerja graph secara keseluruhan

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided (SOAL 1)

("graph.h")

```
#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED

#include <iostream>
using namespace std;

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct ElmEdge *adrEdge;

struct ElmNode {
   infoGraph info;
   int visited;
   adrEdge firstEdge;
   adrNode next;
};

struct ElmEdge {
   adrNode node;
   adrEdge next;
};

struct Graph {
   adrNode first;
};

void createGraph(Graph &G);
void insertNode(Graph &G, infoGraph X);
void connectNode(Graph &G, adrNode N1, adrNode N2);
void printInfoGraph(Graph G);

adrNode findNode(Graph G, infoGraph X);
void addEdge(adrNode NAsal, adrNode NTujuan);
```

```
void resetVisited(Graph &G);
void printDFS(Graph G, adrNode N);
void printBFS(Graph G, adrNode N);

#endif
```

(“graph.cpp”)

```
#include "graph.h"
#include <stack>
#include <queue>

void createGraph(Graph &G) {
   G.first = NULL;
}

void insertNode(Graph &G, infoGraph X) {
   adrNode P = new ElmNode;
   P->info = X;
   P->visited = 0;
   P->firstEdge = NULL;
   P->next = NULL;

   if (G.first == NULL) {
      G.first = P;
   } else {
      adrNode last = G.first;
      while (last->next != NULL) {
         last = last->next;
      }
      last->next = P;
   }
}

adrNode findNode(Graph G, infoGraph X) {
   adrNode P = G.first;
   while (P != NULL) {
      if (P->info == X) {
         return P;
      }
      P = P->next;
   }
   return NULL;
```

```
}

void addEdge(adrNode NAsal, adrNode NTujuan) {
   adrEdge newEdge = new ElmEdge;
   newEdge->node = NTujuan;
   newEdge->next = NULL;

   if (NAsal->firstEdge == NULL) {
      NAsal->firstEdge = newEdge;
   } else {
      adrEdge last = NAsal->firstEdge;
      while (last->next != NULL) {
         last = last->next;
      }
      last->next = newEdge;
   }
}

void connectNode(Graph &G, adrNode N1, adrNode N2) {
   if (N1 != NULL && N2 != NULL) {
      addEdge(N1, N2);
      addEdge(N2, N1);
   }
}

void printInfoGraph(Graph G) {
   adrNode P = G.first;
   while (P != NULL) {
      cout << "Node " << P->info << " terhubung dengan: ";
      adrEdge E = P->firstEdge;
      while (E != NULL) {
         cout << E->node->info << " ";
         E = E->next;
      }
      cout << endl;
      P = P->next;
   }
}

// Implementasi(DFS & BFS)

void resetVisited(Graph &G) {
   adrNode P = G.first;
```

```
    while (P != NULL) {
        P->visited = 0;
        P = P->next;
    }
}

void printDFS(Graph G, adrNode N) {
    if (N == NULL) return;

    resetVisited(G);

    stack<adrNode> S;
    S.push(N);

    cout << "DFS Traversal: ";

    while (!S.empty()) {
        adrNode curr = S.top();
        S.pop();

        if (curr->visited == 0) {
            curr->visited = 1;
            cout << curr->info << " ";

            adrEdge E = curr->firstEdge;
            while (E != NULL) {
                if (E->node->visited == 0) {
                    S.push(E->node);
                }
                E = E->next;
            }
        }
    }
    cout << endl;
}

void printBFS(Graph G, adrNode N) {
    if (N == NULL) return;

    resetVisited(G);

    queue<adrNode> Q;
    Q.push(N);
```

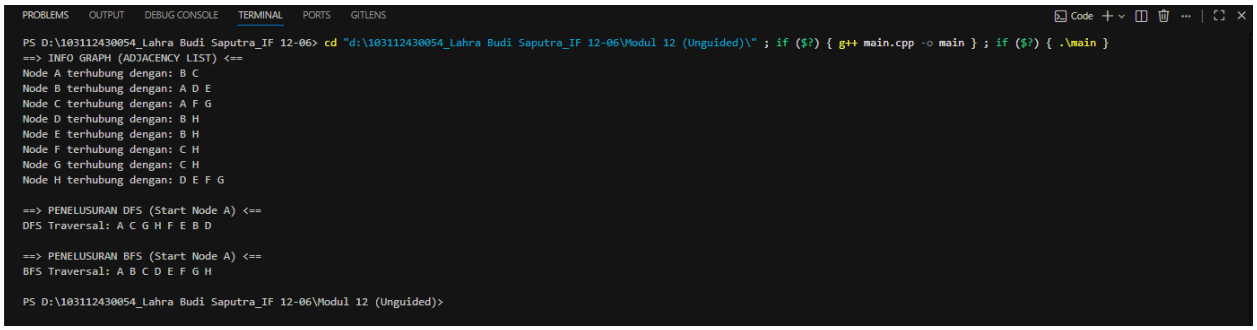
```
    N->visited = 1;

    cout << "BFS Traversal: ";

    while (!Q.empty()) {
        adrNode curr = Q.front();
        Q.pop();
        cout << curr->info << " ";

        adrEdge E = curr->firstEdge;
        while (E != NULL) {
            if (E->node->visited == 0) {
                E->node->visited = 1;
                Q.push(E->node);
            }
            E = E->next;
        }
    }
    cout << endl;
}
```

(“main.cpp”)

```
#include <iostream>
#include "graph.h"
#include "graph.cpp"

using namespace std;

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');

    adrNode A = findNode(G, 'A');
```

```
    adrNode B = findNode(G, 'B');
    adrNode C = findNode(G, 'C');
    adrNode D = findNode(G, 'D');
    adrNode E = findNode(G, 'E');
    adrNode F = findNode(G, 'F');
    adrNode G_Node = findNode(G, 'G');
    adrNode H = findNode(G, 'H');

    connectNode(G, A, B);
    connectNode(G, A, C);

    connectNode(G, B, D);
    connectNode(G, B, E);

    connectNode(G, C, F);
    connectNode(G, C, G_Node);

    connectNode(G, D, H);
    connectNode(G, E, H);
    connectNode(G, F, H);
    connectNode(G, G_Node, H);

    cout << "==> INFO GRAPH (ADJACENCY LIST) <==" << endl;
    printInfoGraph(G);
    cout << endl;

    cout << "==> PENELUSURAN DFS (Start Node A) <==" << endl;
    printDFS(G, A);
    cout << endl;

    cout << "==> PENELUSURAN BFS (Start Node A) <==" << endl;
    printBFS(G, A);
    cout << endl;

    return 0;
}
```

Screenshots Output:



Deskripsi:

Kode program di atas ini merupakan implementasi lengkap dari Abstract Data Type (ADT) Graph Tidak Berarah (Undirected Graph) dengan representasi adjacency list (senarai ketetanggaan). Struktur data ini dibangun melalui tiga file terintegrasi: graph.h yang mendefinisikan tipe data Node (simpul) dan Edge (sisi/penghubung) menggunakan pointer; graph.cpp yang berisi logika manajemen memori, pembentukan koneksi dua arah antar-node, serta algoritma penelusuran; dan main.cpp sebagai penggerak utama. Program ini secara spesifik mereplikasi topologi graf dari Node A hingga H sesuai ilustrasi modul , kemudian memvalidasi strukturnya dengan menampilkan daftar ketetanggaan serta melakukan simulasi penelusuran data menggunakan metode Depth First Search (DFS) berbasis Stack dan Breadth First Search (BFS) berbasis Queue

D. Kesimpulan

Penerapan latihan ini memberikan pemahaman mendalam mengenai transformasi konsep teoritis struktur data non-linear menjadi aplikasi pemrograman dinamis. Melalui implementasi graf, terlihat jelas bagaimana relasi data yang kompleks (banyak-ke-banyak) dapat dikelola secara efisien menggunakan manipulasi *pointer* dan *linked list*. Lebih jauh lagi, integrasi algoritma penelusuran DFS dan BFS menegaskan perbedaan fundamental dalam strategi pencarian data; di mana DFS mengajarkan penelusuran berbasis kedalaman rekursif atau tumpukan, sedangkan BFS mengajarkan penelusuran berbasis level atau antrean. Hal ini menyimpulkan bahwa pemilihan struktur data pendukung (*Stack* vs *Queue*) sangat menentukan perilaku dan hasil urutan pemrosesan data dalam sebuah jaringan

E. Referensi

Elkari, B., et al. (2024). A comparative study of DFS, BFS, and A search algorithms. *International Journal of Computer Science and Mathematical Applications,* 10(1), 1–12. https://pdfs.semanticscholar.org/995d/7805d7454a316dbf6dff9b7bd98348e4667d.pdf

Everitt, T., et al. (2015). *Analytical results on the BFS vs. DFS algorithm selection problem. arXiv.* https://arxiv.org/abs/1509.02709

Munir, R. (2008). *Perbandingan algoritma penelusuran Depth First Search dan Breadth First Search. Makalah Matematika Diskrit,* Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2008-2009/Makalah2008/Makalah0809-054.pdf