LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XII
GRAPH

Disusun Oleh :
NAMA : Afief Amar Purnomo
NIM : 103112430067

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Teori dasar yang umum digunakan: Graf adalah struktur data abstrak yang merepresentasikan sekumpulan simpul (vertex/node) dan himpunan sisi (edge) yang menghubungkan pasangan simpul, dapat berarah (directed) maupun tak berarah (undirected), berbobot atau tidak berbobot, dan sangat cocok untuk memodelkan berbagai permasalahan relasi seperti jaringan jalan, jaringan komputer, atau hubungan pertemanan dalam media sosial.

Dalam C++ graf biasanya direpresentasikan dengan matriks ketetanggaan (array dua dimensi) atau daftar ketetanggaan (list/multilist) menggunakan pointer dan struktur data dinamis sehingga penambahan dan penghapusan simpul serta sisi dapat dilakukan secara fleksibel, kemudian algoritma penelusuran seperti Breadth First Search (BFS) dengan bantuan queue dan Depth First Search (DFS) dengan bantuan stack atau rekursi dimanfaatkan untuk menjelajahi vertex, mencari jalur, atau menguji keterhubungan graf, sementara algoritma lanjutan seperti Dijkstra, Prim, dan Kruskal memanfaatkan representasi graf tersebut untuk menyelesaikan masalah jalur terpendek dan pembentukan minimum spanning tree secara efisien dalam program C++.

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

Guided 1

(file graf.h)

graf.h > PrintDFS(Graph &, adrNode)

```
1   #ifndef GRAF_H_INCLUDED
2   #define GRAF_H_INCLUDED
3
4   #include <iostream>
5   using namespace std;
6
7   typedef char infoGraph;
8
9   struct ElmNode;
10  struct ElmEdge;
11
12  typedef ElmNode *adrNode;
13  typedef ElmEdge *adrEdge;
14
15  struct ElmNode
16  {
17      infoGraph info;
18      int visited;
19      adrEdge firstEdge;
20      adrNode next;
21  };
22
23  struct ElmEdge
24  {
25      adrNode node;
26      adrEdge next;
27  };
28
29  struct Graph
30  {
31      adrNode first;
32  };
```

```
33
34  //PRIMITIF GRAPH
    Tabnine | Edit | Test | Explain | Document
35  void CreateGraph(Graph &G);
    Tabnine | Edit | Test | Explain | Document
36  adrNode AllocateNode(infoGraph X);
    Tabnine | Edit | Test | Explain | Document
37  adrEdge AllocateEdge(adrNode N);
38
    Tabnine | Edit | Test | Explain | Document
39  void InsertNode(Graph &G, infoGraph X);
    Tabnine | Edit | Test | Explain | Document
40  adrNode FindNode(Graph G, infoGraph X);
41
    Tabnine | Edit | Test | Explain | Document
42  void ConnectNode(Graph &G, infoGraph A, infoGraph B);
43
    Tabnine | Edit | Test | Explain | Document
44  void PrintInfoGraph(Graph G);
45
46  //Traversal
    Tabnine | Edit | Test | Explain | Document
47  void ResetVisited(Graph &G);
    Tabnine | Edit | Test | Explain | Document
48  void PrintDFS(Graph &G, adrNode N);
    Tabnine | Edit | Test | Explain | Document
49  void PrintBFS(Graph &G, adrNode N);
50
51  #endif
```

(file graf.cpp)

graf.cpp > PrintDFS(Graph &, adrNode)

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

Tabnine | Edit | Test | Explain | Document
void CreateGraph(Graph &G)
{
    G.first = NULL;
}

Tabnine | Edit | Test | Explain | Document
adrNode AllocateNode(infoGraph X)
{
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}

Tabnine | Edit | Test | Explain | Document
adrEdge AllocateEdge(adrNode N)
{
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}

Tabnine | Edit | Test | Explain | Document
void InsertNode (Graph &G, infoGraph X)
```

```
29  {
30      adrNode P = AllocateNode(X);
31      P->next = G.first;
32      G.first = P;
33  }
34
    Tabnine | Edit | Test | Explain | Document
35  adrNode FindNode(Graph G, infoGraph X)
36  {
37      adrNode P = G.first;
38      while (P != NULL)
39      {
40          if(P->info == X)
41              return P;
42          P = P->next;
43      }
44      return NULL;
45  }
46
    Tabnine | Edit | Test | Explain | Document
47  void ConnectNode(Graph &G, infoGraph A, infoGraph B)
48  {
49      adrNode N1 = FindNode(G, A);
50      adrNode N2 = FindNode(G, B);
51
52      if (N1 == NULL || N2 == NULL)
53      {
54          cout << "Node tidak ditemukan\n";
55          return;
56      }
57
```

```
    //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;
}

Tabnine | Edit | Test | Explain | Document
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;
    }
}

Tabnine | Edit | Test | Explain | Document
void ResetVisited(Graph &G)
```

```
87  {
88      adrNode P = G.first;
89      while (P != NULL)
90      {
91          P->visited = 0;
92          P = P->next;
93      }
94  }
95
    Tabnine | Edit | Test | Explain | Document
96  void PrintDFS(Graph &G, adrNode N)
97  {
98      if (N == NULL)
99          return;
100
101     N->visited = 1;
102     cout << N->info << " ";
103     adrEdge E = N->firstEdge;
104
105     while (E != NULL)
106     {
107         if (E->node->visited == 0)
108         {
109             PrintDFS(G, E->node);
110         }
111         E = E->next;
112     }
113 }
114
    Tabnine | Edit | Test | Explain | Document
115 void PrintBFS(Graph &G, adrNode N)
```

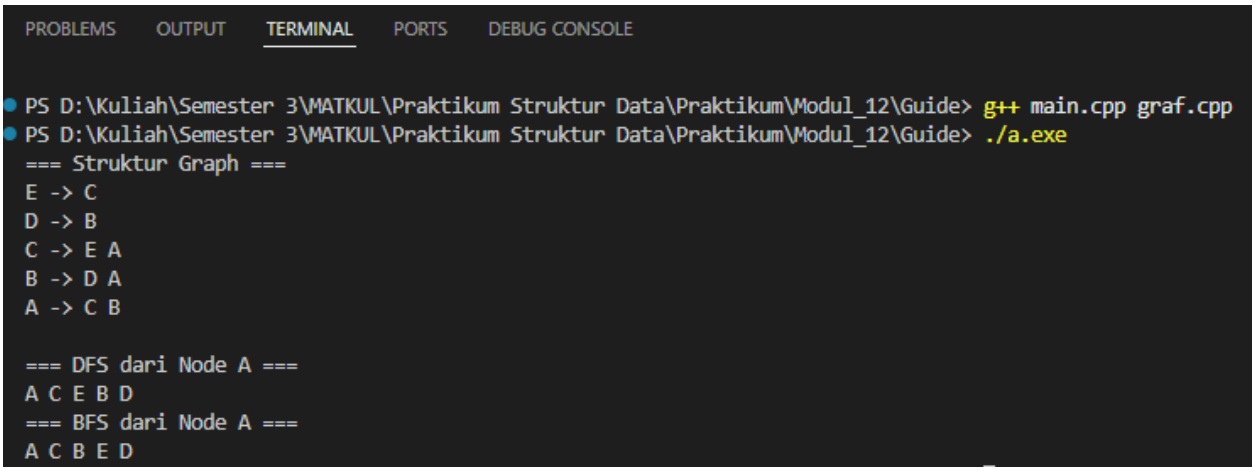
```
{
    if(N == NULL)
        return;

    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;
            }
        }
    }
}
```

(file main.cpp)

main.cpp > main()

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

Tabnine | Edit | Test | Explain | Document
int main()
{
    Graph G;
    CreateGraph(G);

    //Tambah node
    InsertNode(G, 'A'); //0
    InsertNode(G, 'B'); //1
    InsertNode(G, 'C'); //2
    InsertNode(G, 'D'); //3
    InsertNode(G, 'E'); //4

    //Tambah edge
    ConnectNode(G, 'A', 'B'); //0 -> 1
    ConnectNode(G, 'A', 'C'); //0 -> 2
    ConnectNode(G, 'B', 'D'); //1 -> 3
    ConnectNode(G, 'C', 'E'); //2 -> 4

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

    cout << "\n=== DFS dari Node A ===\n";
    ResetVisited(G); //Reset visited semua node
    PrintDFS(G, FindNode(G, 'A'));

    cout << "\n=== BFS dari Node A ===\n";
    ResetVisited(G); //Reset visited semua node
    PrintBFS(G, FindNode(G, 'A'));

    cout << endl;
    return 0;
}
```

Screenshots Output



Deskripsi:

Program di atas merupakan implementasi struktur data graph menggunakan adjacency list dengan linked list sebagai representasi node dan edge, di mana setiap node (vertex) dapat memiliki banyak edge (sisi) yang menghubungkannya ke node lain. Graph disimpan dalam sebuah linked list node (Graph.first), dan setiap node memiliki linked list edge (firstEdge) yang menunjuk ke node tujuan. Program menyediakan operasi dasar seperti CreateGraph untuk inisialisasi graph, InsertNode untuk menambahkan node baru, ConnectNode untuk menghubungkan dua node (pada kode ini bersifat undirected karena edge dibuat dua arah), serta PrintInfoGraph untuk menampilkan struktur graph beserta relasi antar node. Selain itu, program juga mendukung traversal graph menggunakan Depth First Search (DFS) melalui fungsi PrintDFS dan Breadth First Search (BFS) melalui fungsi PrintBFS, dengan bantuan penanda visited untuk mencegah kunjungan ulang. Pada fungsi main, graph dibangun dengan lima node yaitu A, B, C, D, dan E, kemudian dihubungkan sehingga A terhubung ke B dan C, B ke D, serta C ke E. Setelah struktur graph ditampilkan, program menelusuri graph mulai dari node A menggunakan DFS dan BFS untuk memperlihatkan urutan kunjungan node berdasarkan kedua metode traversal tersebut.

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

Unguided 1 (soal 1)

(file graph.h)

graph.h > PrintBFS(Graph &, adrNode)

```
1   #ifndef GRAPH_H
2   #define GRAPH_H
3
4   #include <iostream>
5   using namespace std;
6
7   typedef char infograph;
8   typedef struct ElmNode* adrNode;
9   typedef struct ElmEdge* adrEdge;
10
11  struct ElmEdge {
12      adrNode nextNode;
13      adrEdge nextEdge;
14  };
15
16  struct ElmNode {
17      infograph info;
18      int visited;
19      adrEdge firstEdge;
20      adrNode nextNode;
21  };
22
23  struct Graph {
24      adrNode first;
25  };
26
    Tabnine | Edit | Test | Explain | Document
27  void CreateGraph(Graph &G);
    Tabnine | Edit | Test | Explain | Document
28  adrNode InsertNode(Graph &G, infograph x);
    Tabnine | Edit | Test | Explain | Document
29  void ConnectNode(adrNode from, adrNode to);
    Tabnine | Edit | Test | Explain | Document
30  void PrintInfoGraph(Graph G);
31
    Tabnine | Edit | Test | Explain | Document
32  void PrintDFS(Graph &G, adrNode N);
    Tabnine | Edit | Test | Explain | Document
33  void PrintBFS(Graph &G, adrNode N);
34
35  #endif
36
```

(file graph.cpp)

graph.cpp > ...

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

Tabnine | Edit | Test | Explain | Document
void CreateGraph(Graph &G) {
    G.first = NULL;
}

Tabnine | Edit | Test | Explain | Document
adrNode InsertNode(Graph &G, infograph x) {
    adrNode N = new ElmNode;
    N->info = x;
    N->visited = 0;
    N->firstEdge = NULL;
    N->nextNode = G.first;
    G.first = N;
    return N;
}

Tabnine | Edit | Test | Explain | Document
void ConnectNode(adrNode from, adrNode to) {
    adrEdge E = new ElmEdge;
    E->nextNode = to;
    E->nextEdge = from->firstEdge;
    from->firstEdge = E;
}

Tabnine | Edit | Test | Explain | Document
void PrintInfoGraph(Graph G) {
    adrNode N = G.first;
    while (N != NULL) {
        cout << N->info << " -> ";
```

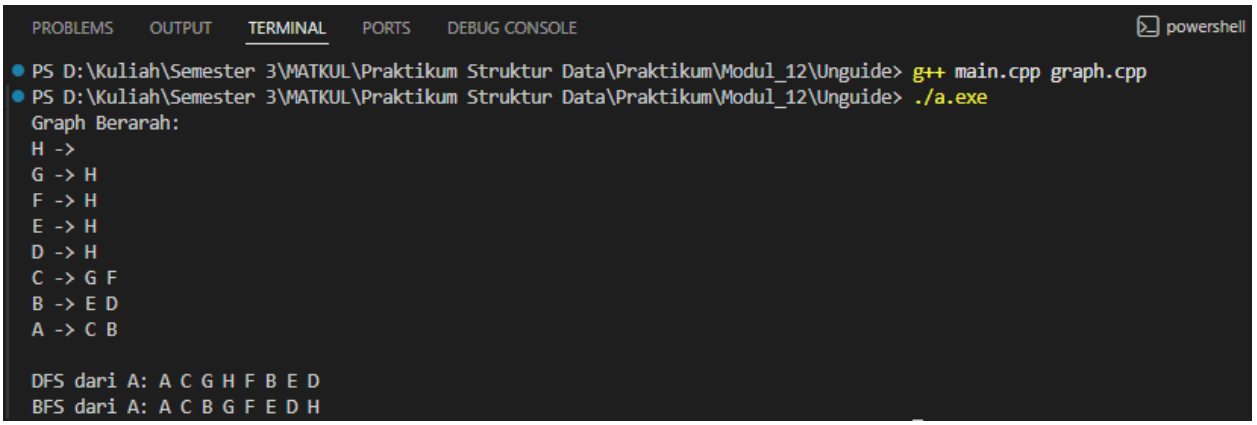
```
        adrEdge E = N->firstEdge;
        while (E != NULL) {
            cout << E->nextNode->info << " ";
            E = E->nextEdge;
        }
        cout << endl;
        N = N->nextNode;
    }
}

Tabnine | Edit | Test | Explain | Document
void PrintDFS(Graph &G, adrNode N) {
    if (N == NULL || N->visited == 1)
        return;

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

    adrEdge E = N->firstEdge;
    while (E != NULL) {
        PrintDFS(G, E->nextNode);
        E = E->nextEdge;
    }
}

Tabnine | Edit | Test | Explain | Document
void PrintBFS(Graph &G, adrNode start) {
    queue<adrNode> Q;

    start->visited = 1;
    Q.push(start);

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

        adrEdge E = N->firstEdge;
        while (E != NULL) {
            if (E->nextNode->visited == 0) {
                E->nextNode->visited = 1;
                Q.push(E->nextNode);
            }
            E = E->nextEdge;
        }
    }
}

```

(file main.cpp)

main.cpp > main()

```cpp
#include "graph.h"

Tabnine | Edit | Test | Explain | Document
int main() {
    Graph G;
    CreateGraph(G);

    adrNode A = InsertNode(G, 'A');
    adrNode B = InsertNode(G, 'B');
    adrNode C = InsertNode(G, 'C');
    adrNode D = InsertNode(G, 'D');
    adrNode E = InsertNode(G, 'E');
    adrNode F = InsertNode(G, 'F');
    adrNode Gg = InsertNode(G, 'G');
    adrNode H = InsertNode(G, 'H');

    ConnectNode(A, B);
    ConnectNode(A, C);

    ConnectNode(B, D);
    ConnectNode(B, E);

    ConnectNode(C, F);
    ConnectNode(C, Gg);

    ConnectNode(D, H);
    ConnectNode(E, H);
    ConnectNode(F, H);
    ConnectNode(Gg, H);

    cout << "Graph Berarah:\n";
    PrintInfoGraph(G);

    cout << "\nDFS dari A: ";
    PrintDFS(G, A);

    adrNode N = G.first;
    while (N != NULL) {
        N->visited = 0;
        N = N->nextNode;
    }

    cout << "\nBFS dari A: ";
    PrintBFS(G, A);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program di atas merupakan implementasi struktur data graph berarah (directed graph) yang direpresentasikan menggunakan adjacency list berbasis linked list, di mana setiap node (vertex) disimpan dalam linked list utama dan setiap node memiliki linked list edge yang menunjuk ke node tujuan. Struktur ElmNode menyimpan informasi node, penanda visited untuk traversal, serta pointer ke edge pertama dan node berikutnya, sedangkan ElmEdge menyimpan hubungan berarah ke node lain. Program menyediakan operasi CreateGraph untuk inisialisasi graph, InsertNode untuk menambahkan node baru, ConnectNode untuk membuat sisi berarah dari satu node ke node lain, dan PrintInfoGraph untuk menampilkan struktur graph beserta arah keterhubungannya. Selain itu, program mendukung penelusuran graph menggunakan Depth First Search (DFS) dan Breadth First Search (BFS) melalui fungsi PrintDFS dan PrintBFS dengan memanfaatkan atribut visited agar node tidak dikunjungi lebih dari satu kali. Pada fungsi main, graph dibangun dengan delapan node yaitu A hingga H, kemudian dihubungkan secara berarah sehingga membentuk struktur bertingkat yang berujung pada node H. Setelah struktur graph ditampilkan, program melakukan traversal DFS dan BFS dimulai dari node A untuk memperlihatkan perbedaan urutan kunjungan node pada kedua metode penelusuran tersebut.

D. Kesimpulan

Kesimpulan dari Praktikum Modul XII tentang Graph adalah bahwa graph merupakan struktur data yang digunakan untuk merepresentasikan hubungan antar objek dalam bentuk simpul (vertex) dan sisi (edge), baik berarah maupun tidak berarah, yang sangat efektif untuk memodelkan relasi kompleks. Pada praktikum ini, graph diimplementasikan menggunakan adjacency list berbasis multi linked list dengan bahasa C++, sehingga setiap node dapat memiliki banyak edge yang terhubung ke node lain. Mahasiswa mempelajari operasi dasar graph seperti pembuatan graph, penambahan node, penghubungan antar node, serta penelusuran graph menggunakan algoritma Depth First Search (DFS) dan Breadth First Search (BFS). Sebagai contoh, ketika node A dihubungkan dengan node B dan C, lalu B dihubungkan dengan D dan C dengan E, maka traversal DFS akan menelusuri graph secara mendalam hingga ke node terakhir sebelum kembali, sedangkan BFS akan menelusuri graph berdasarkan level kedekatan node. Selain itu, pada graph berarah, arah edge menentukan jalur penelusuran sehingga hubungan antar node tidak selalu bersifat dua arah. Secara keseluruhan, praktikum ini membantu mahasiswa memahami konsep graph, representasinya menggunakan pointer, serta penerapannya dalam kasus nyata seperti jaringan komputer, peta jalan, dan sistem relasi data yang membutuhkan penelusuran dan analisis keterhubungan antar objek.

E. Referensi

GeeksforGeeks. (2024). Implementation of graph in C++. https://www.geeksforgeeks.org/cpp/implementation-of-graph-in-cpp/

Maulana, F., & Hidayati, R. (2022). Simulasi graf berarah C++ dengan SDL. Jurnal Teknologi Informasi, 8(3), 177–186. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2008-2009/Makalah2008/Makalah0809-097.pdf

Yuliana, S., & Fadli, R. (2023). Implementasi grafik dalam C++ menggunakan daftar ketetanggaan. https://translate.google.com/translate?u=https%3A%2F%2Fwww.softwaretestinghelp.com%2Fgraph-implementation-cpp%2F&hl=id