LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIV
GRAPH

Disusun Oleh :

NAMA : Azzahra Farelika Esti Ning Tyas

NIM : 103112430023

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Graph adalh struktur data non-linier yang digunakan untuk merepresentasikan hubungan antar objek. Secara umum, graph terdiri dari simpul (vertex/node) dan sisi (edge) yang menghubungkan antar simpul. Graph banyak digunakan dalam berbagai bidang seperti jaringan komputer, pemetaan jalan, struktur organisasi, dan penjadwalan proses.

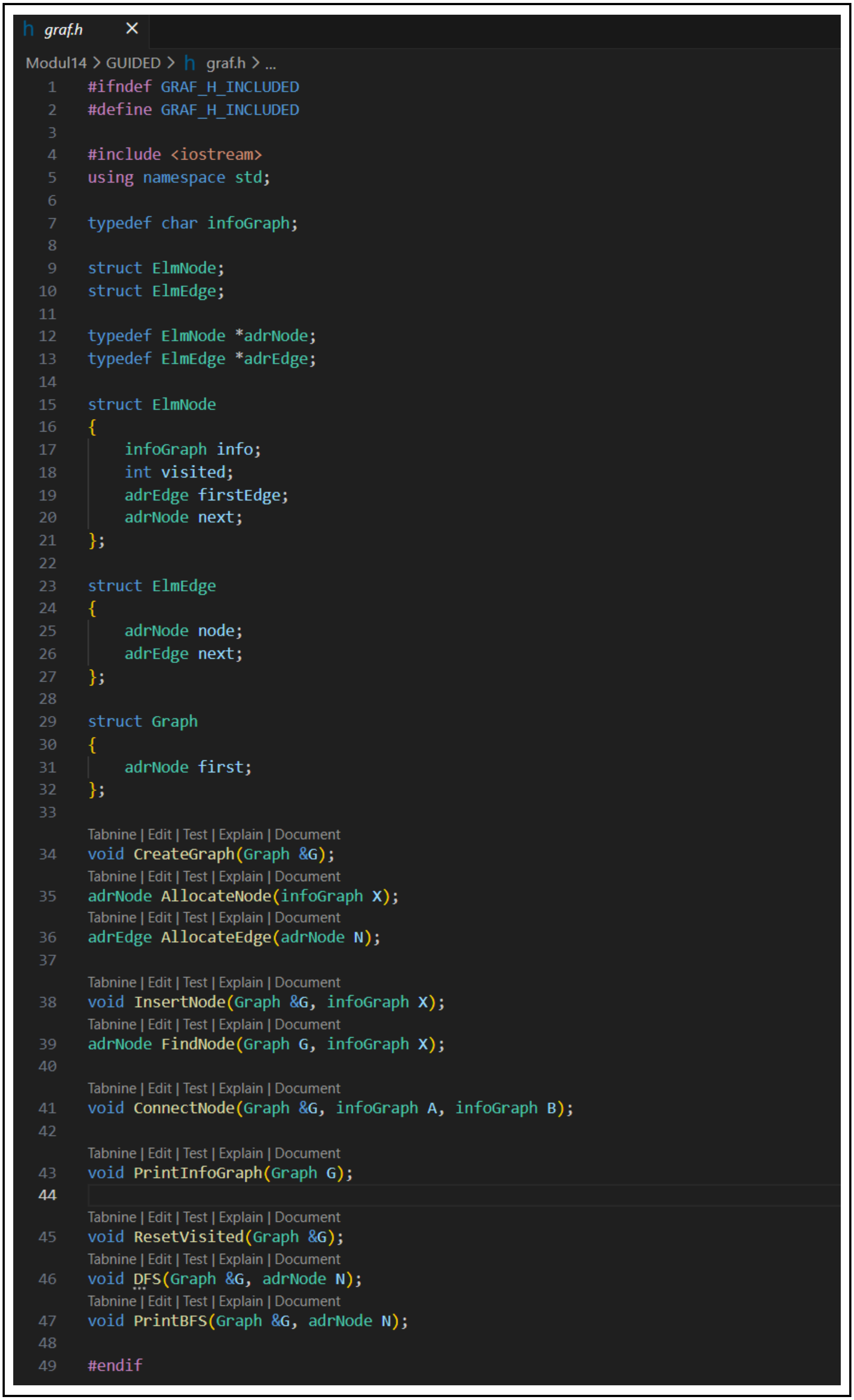
Berdasarkan arah edge-nya, graph dibedakan menjadi graph berarah (directed graph) dan graph tidak berarah (undirected graph). Pada graph berarah, edge memiliki arah tertentu dari satu node ke node lain, sedangkan pada graph tidak berarah edge tidak memiliki arah sehingga hubungan bersifat dua arah.

Graph juga dapat direpresentasikan dalam beberapa bentuk, di antaranya adjacency matrix dan adjacency list (multilist). Adjacency matrix menggunakan matriks dua dimensi untuk menunjukkan keterhubungan antar node, sedangkan adjacency list menyimpan daftar tetangga dari setiap node dan lebih efisien untuk graph yang bersifat dinamis.

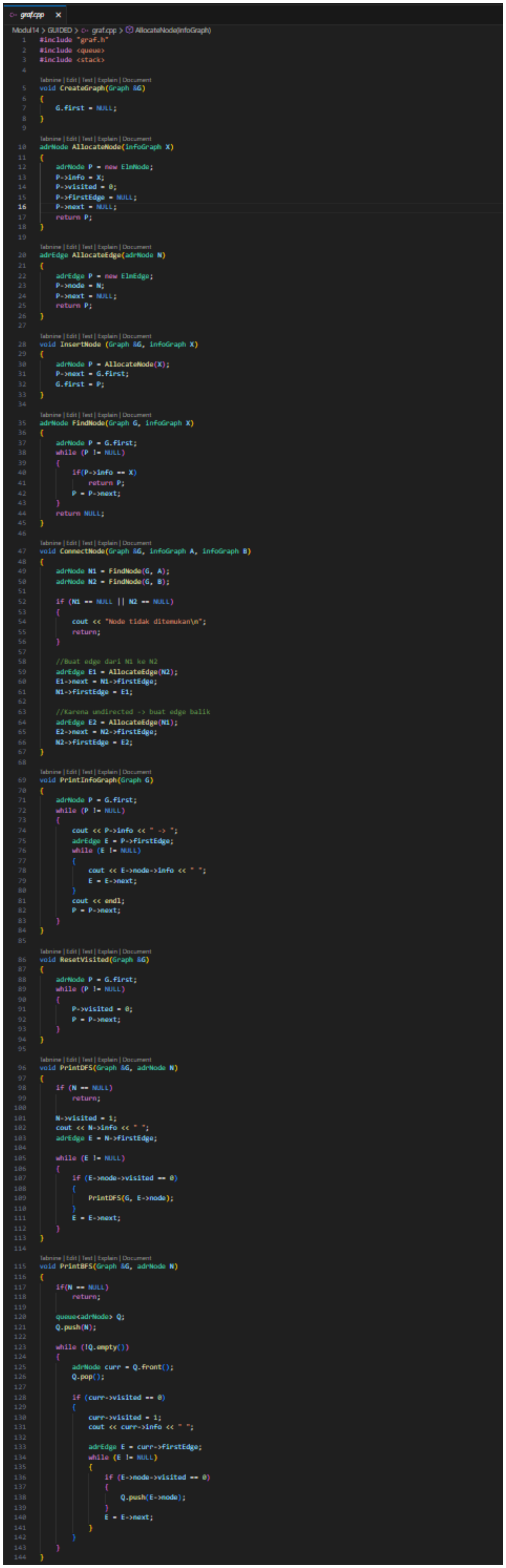
Selain itu, terdapat metode penelusuran graph yang umum digunakan, yaitu Breadth First Search (BFS) dan Depth First Search (DFS). BFS menelusuri graph berdasarkan level menggunakan struktur data queue, sedangkan DFS menelusuri graph secara mendalam menggunakan stack atau rekursi.

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

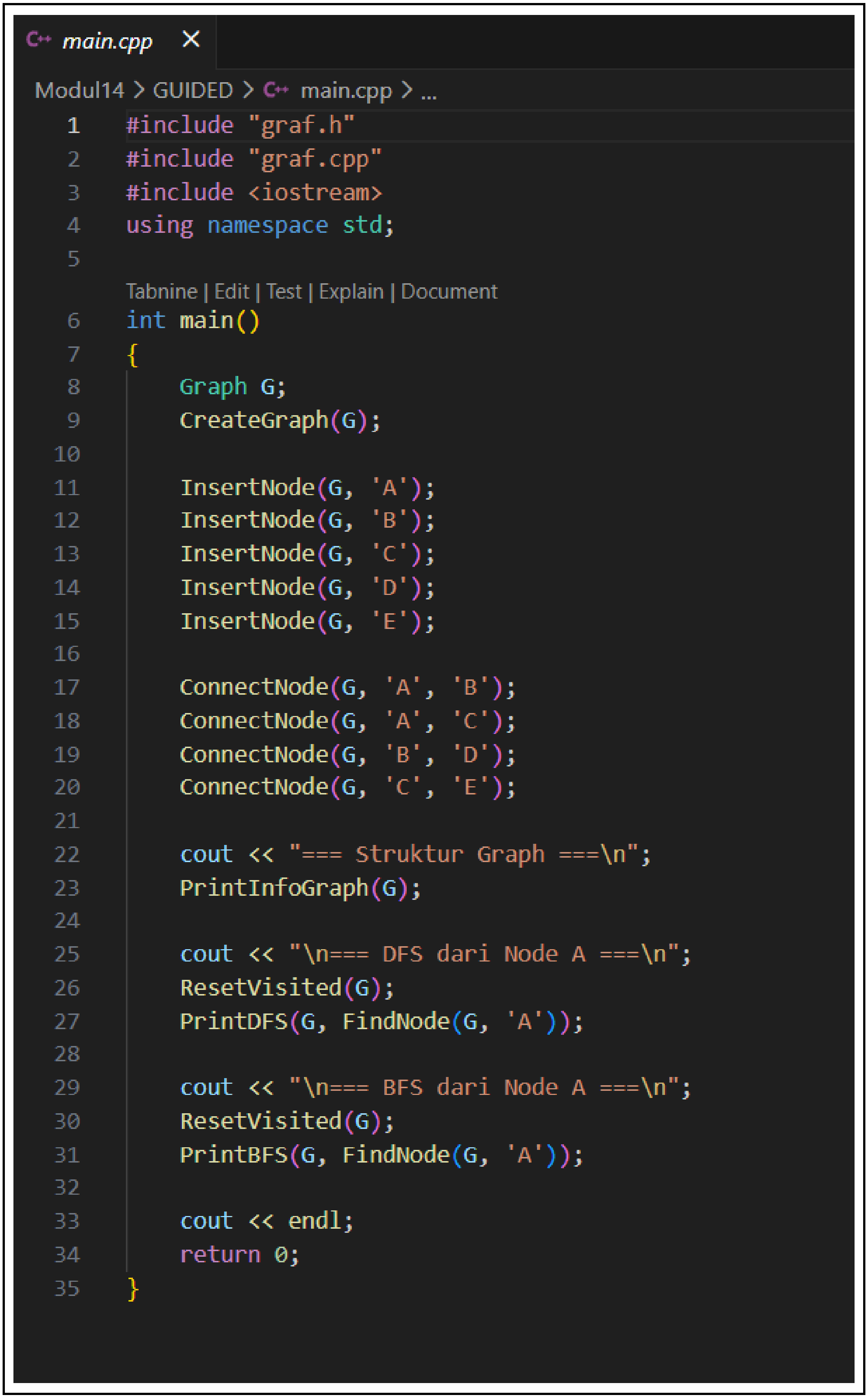
Guided 1


graf.h
Modul14 > GUIDED > graf.h > ...
Tabnine | Edit | Test | Explain | Document

```cpp
#ifndef GRAF_H_INCLUDED
#define GRAF_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;
};

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);

void ResetVisited(Graph &G);
void DFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```


graf.cpp
Modul14 > GUIDED > graf.cpp > AllocateNode(infoGraph)
Tabnine | Edit | Test | Explain | Document

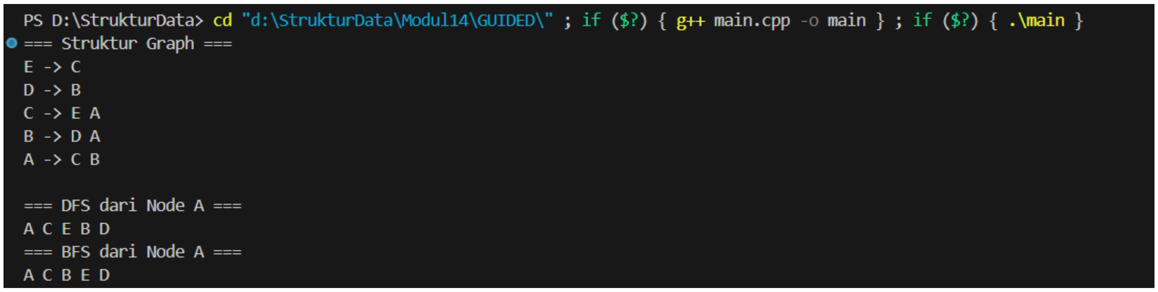
```cpp
#include "graf.h"
#include <queue>
#include <stack>

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 ×
Modul14 > GUIDED > main.cpp > ...
Tabnine | Edit | Test | Explain | Document

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

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

    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');

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

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

Screenshots Output



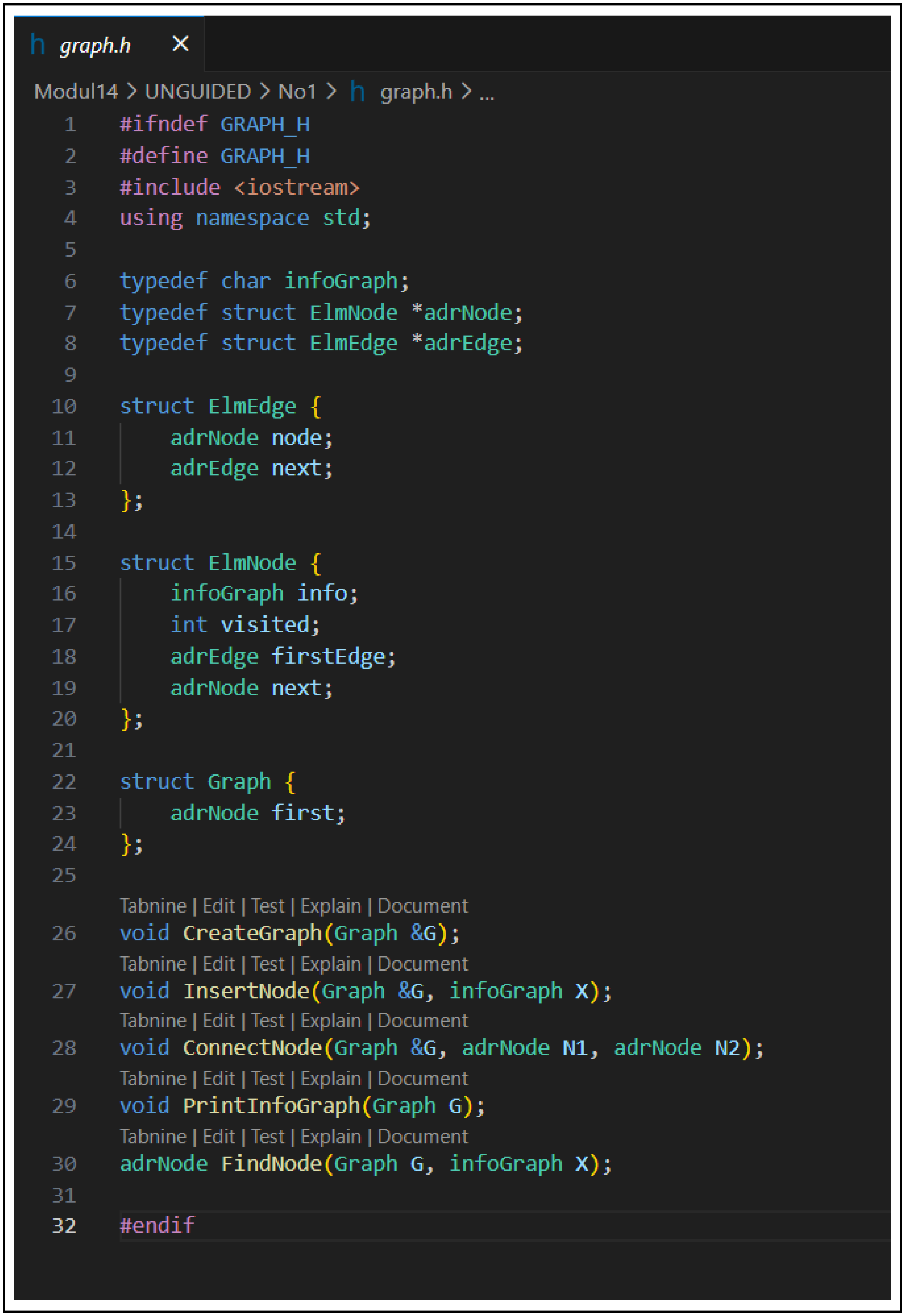
```
PS D:\StrukturData> cd "d:\StrukturData\Modul14\GUIDED\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
=== Struktur Graph ===
E -> C
D -> B
C -> E A
B -> D A
A -> C B

=== DFS dari Node A ===
A C E B D
=== BFS dari Node A ===
A C B E D
```

Deskripsi:

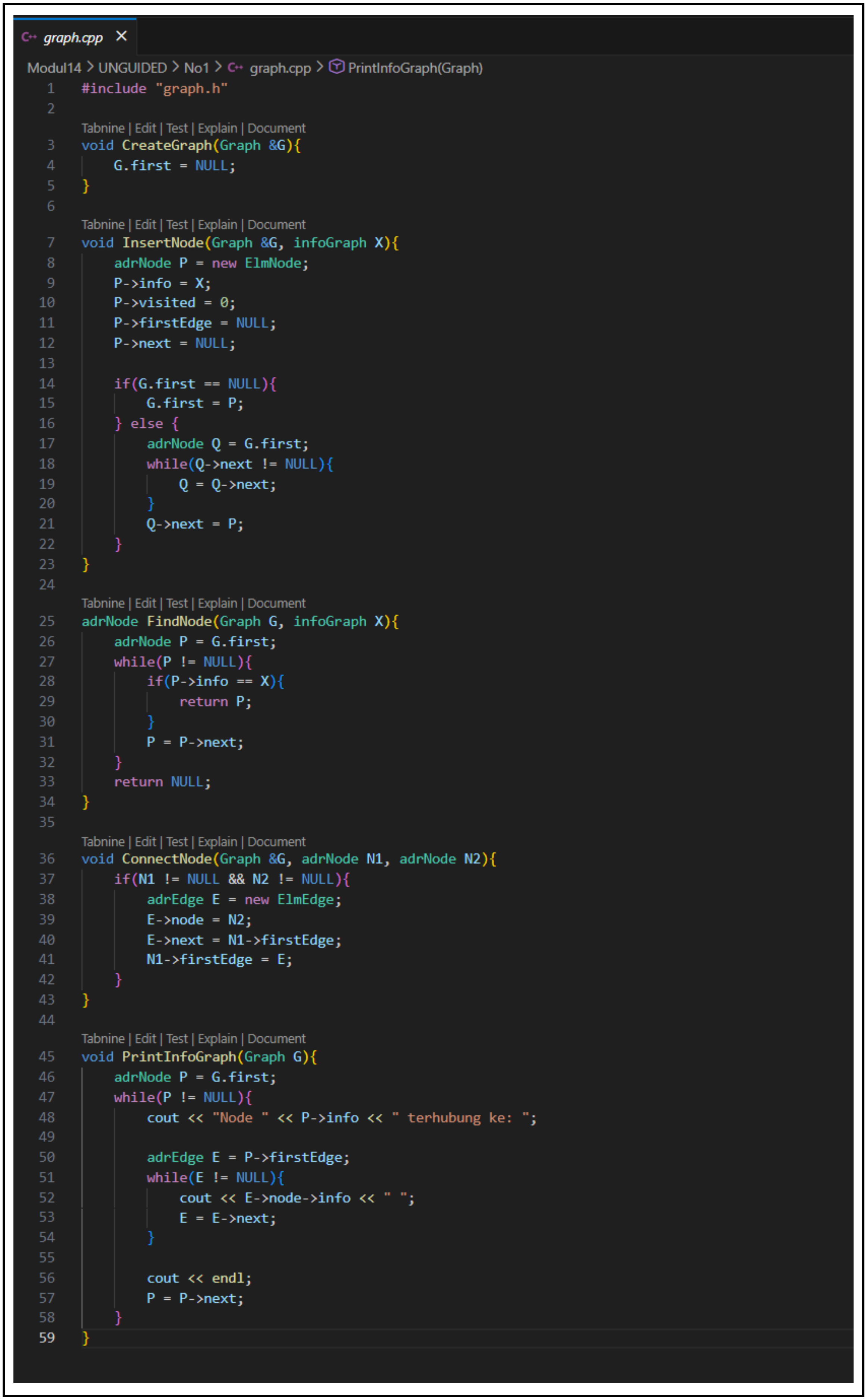
Program ini mengimplementasikan struktur data graph tidak berarah menggunakan representasi adjacency list dengan pointer. File graf.h mendefinisikan struktur node, edge, graph, serta deklarasi fungsi-fungsi dasar graph. File graf.cpp berisi implementasi pembuatan graph, penambahan node, penghubungan antar node, penampilan struktur graph, serta penelusuran Depth First Search (DFS) dan Breadth First Search (BFS). Pada main.cpp, graph dibuat dengan beberapa node dan edge, kemudian ditampilkan strukturnya serta hasil penelusuran DFS dan BFS yang dimulai dari node A.

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

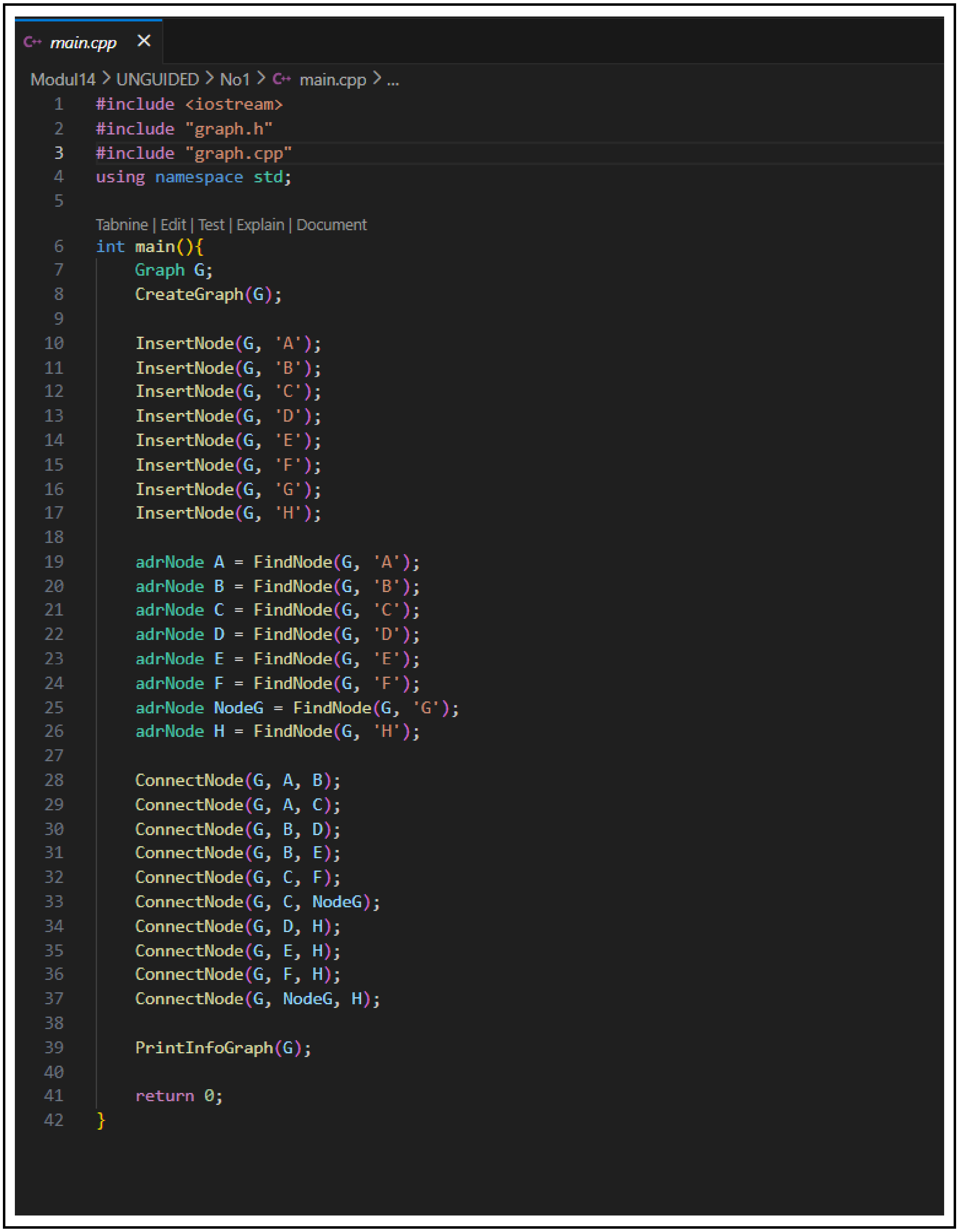
Unguided 1


graph.h
Modul14 > UNGUIDED > No1 > graph.h > ...
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document

```cpp
#ifndef GRAPH_H
#define GRAPH_H
#include <iostream>
using namespace std;

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

struct ElmEdge {
    adrNode node;
    adrEdge next;
};

struct ElmNode {
    infoGraph info;
    int visited;
    adrEdge firstEdge;
    adrNode 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);

#endif
```

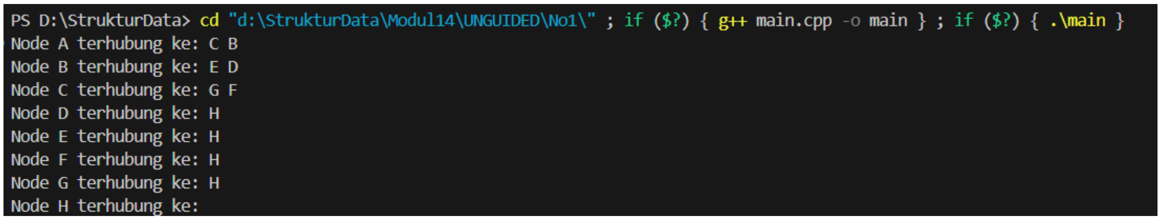

graph.cpp ×
Modul14 > UNGUIDED > No1 > graph.cpp > PrintInfoGraph(Graph)
Tabnine | Edit | Test | Explain | Document

```cpp
#include "graph.h"

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 Q = G.first;
        while(Q->next != NULL){
            Q = Q->next;
        }
        Q->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 ConnectNode(Graph &G, adrNode N1, adrNode N2){
    if(N1 != NULL && N2 != NULL){
        adrEdge E = new ElmEdge;
        E->node = N2;
        E->next = N1->firstEdge;
        N1->firstEdge = E;
    }
}

void PrintInfoGraph(Graph G){
    adrNode P = G.first;
    while(P != NULL){
        cout << "Node " << P->info << " terhubung ke: ";

        adrEdge E = P->firstEdge;
        while(E != NULL){
            cout << E->node->info << " ";
            E = E->next;
        }

        cout << endl;
        P = P->next;
    }
}
```


main.cpp
Modul14 > UNGUIDED > No1 > main.cpp > ...
Tabnine | Edit | Test | Explain | Document

```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 NodeG = 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, NodeG);
    ConnectNode(G, D, H);
    ConnectNode(G, E, H);
    ConnectNode(G, F, H);
    ConnectNode(G, NodeG, H);

    PrintInfoGraph(G);

    return 0;
}
```

Screenshots Output

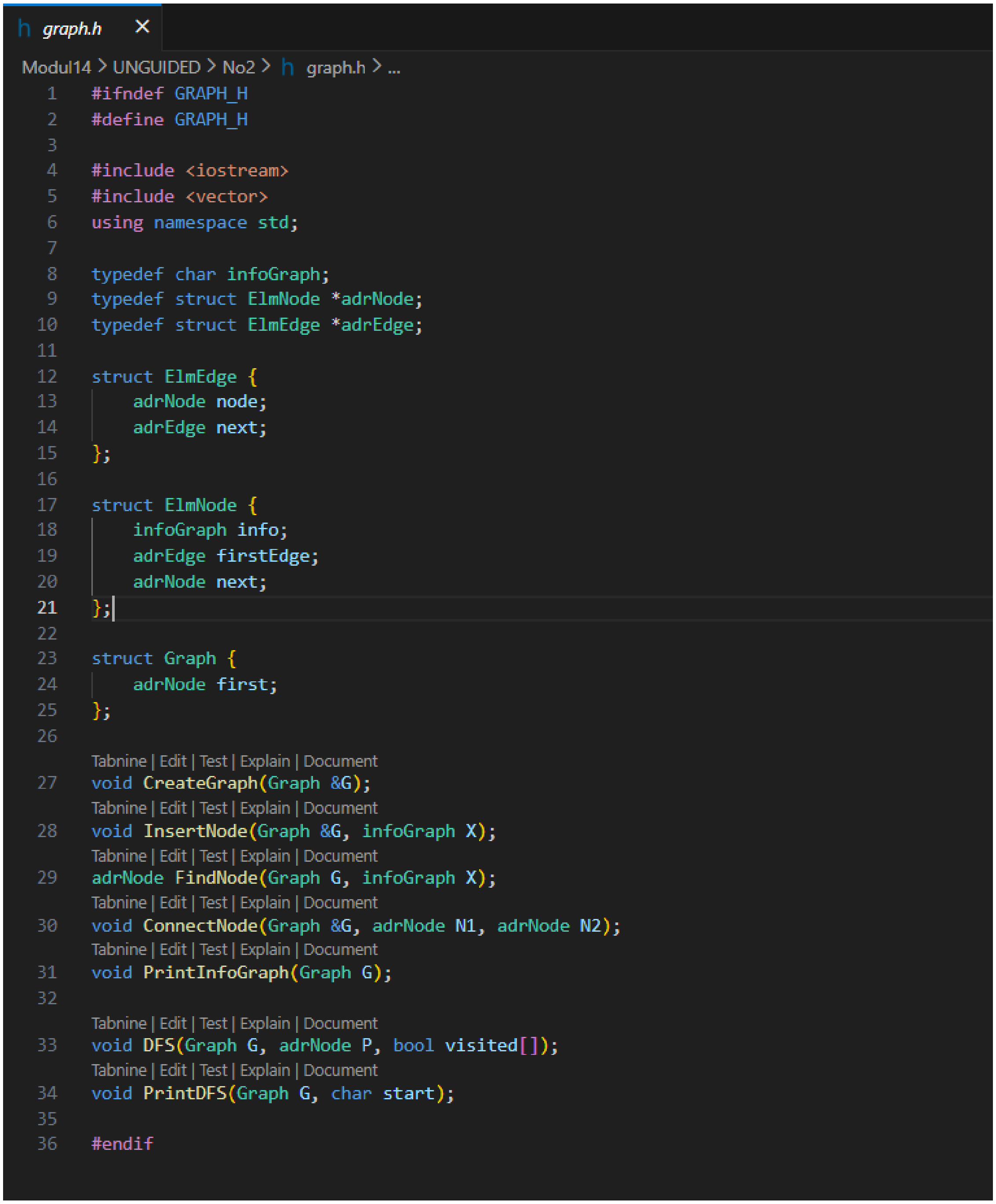


```
PS D:\StrukturData> cd "d:\StrukturData\Modul14\UNGUIDED\No1\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Node A terhubung ke: C B
Node B terhubung ke: E D
Node C terhubung ke: G F
Node D terhubung ke: H
Node E terhubung ke: H
Node F terhubung ke: H
Node G terhubung ke: H
Node H terhubung ke:
```

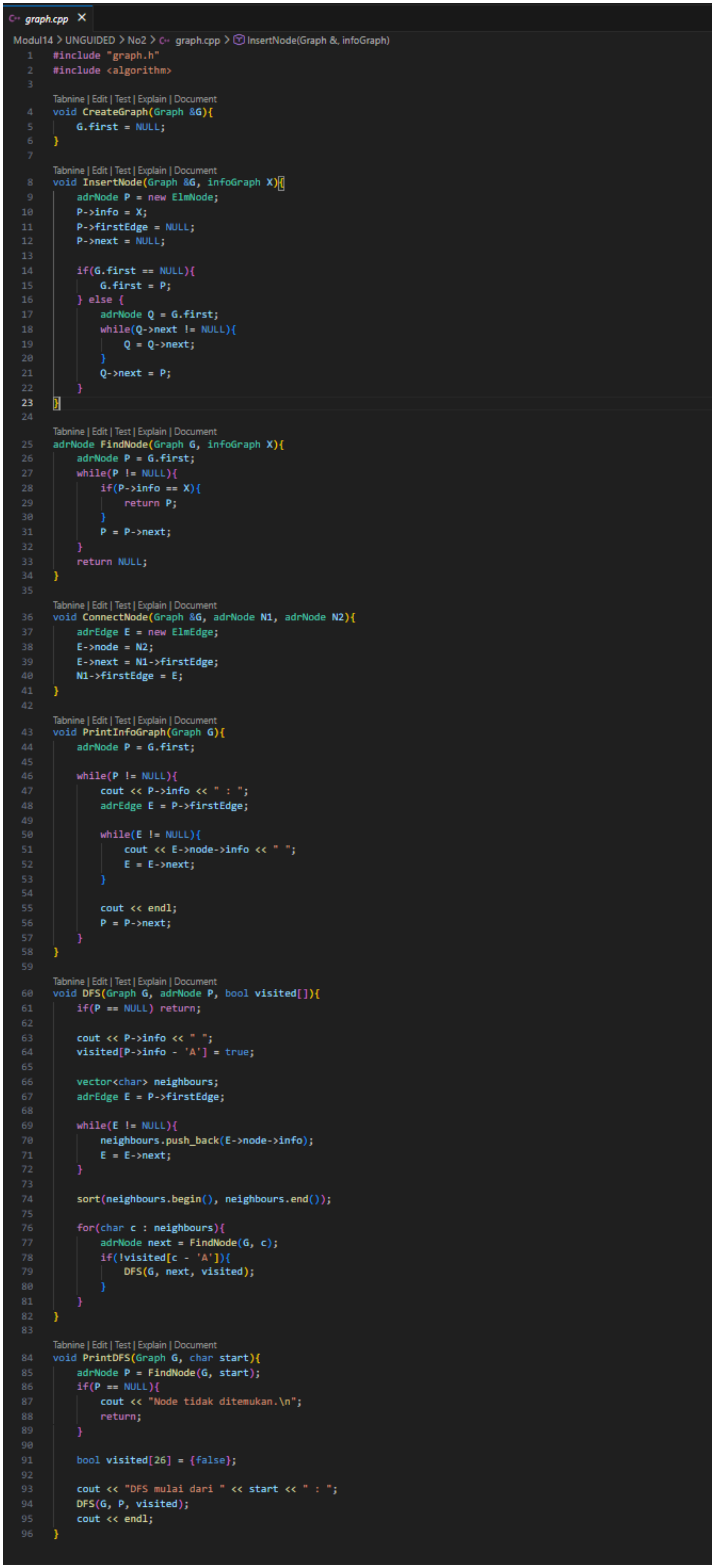
Deskripsi:

Program ini merupakan implementasi graph tidak berarah menggunakan adjacency list. File graph.h berisi definisi struktur data graph, node, dan edge beserta deklarasi fungsi-fungsi dasarnya. File graph.cpp mengimplementasikan operasi graph seperti pembuatan graph, penambahan node, penghubungan antar node, penampilan struktur graph, serta penelusuran menggunakan algoritma Depth First Search (DFS) dan Breadth First Search (BFS). Pada file main.cpp, graph dibangun dengan beberapa node dan edge, kemudian program menampilkan struktur graph serta hasil penelusuran DFS dan BFS yang dimulai dari node tertentu.

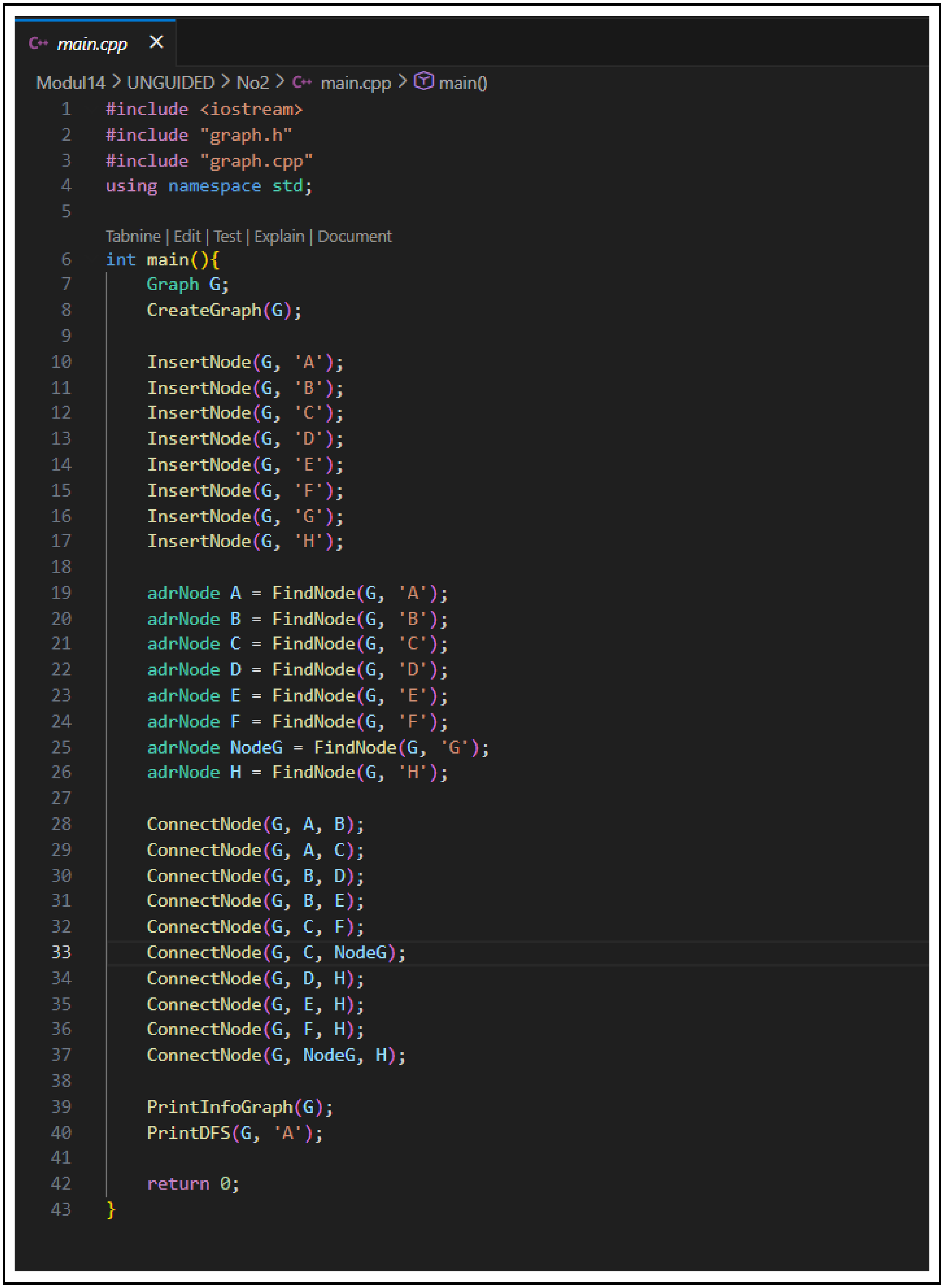
Unguided 2


graph.h
Modul14 > UNGUIDED > No2 > graph.h > ...
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document

```cpp
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
#include <vector>
using namespace std;

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

struct ElmEdge {
    adrNode node;
    adrEdge next;
};

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

struct Graph {
    adrNode first;
};

void CreateGraph(Graph &G);
void InsertNode(Graph &G, infoGraph X);
adrNode FindNode(Graph G, infoGraph X);
void ConnectNode(Graph &G, adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);

void DFS(Graph G, adrNode P, bool visited[]);
void PrintDFS(Graph G, char start);

#endif
```

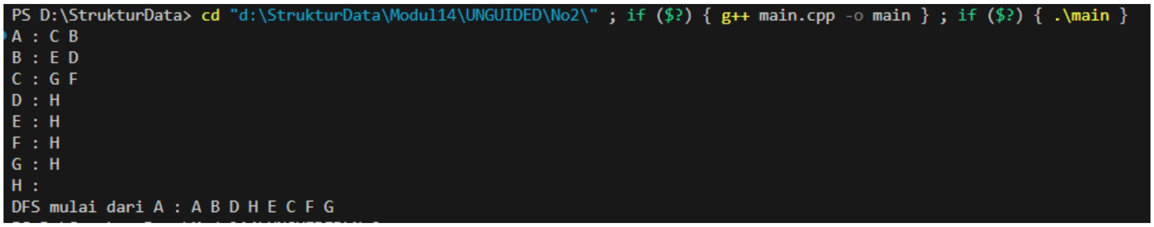

graph.cpp
Modul14 > UNGUIDED > No2 > graph.cpp > InsertNode(Graph &, infoGraph)
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Explain | Document

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

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

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

    if(G.first == NULL){
        G.first = P;
    } else {
        adrNode Q = G.first;
        while(Q->next != NULL){
            Q = Q->next;
        }
        Q->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 ConnectNode(Graph &G, adrNode N1, adrNode N2){
    adrEdge E = new ElmEdge;
    E->node = N2;
    E->next = N1->firstEdge;
    N1->firstEdge = E;
}

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 DFS(Graph G, adrNode P, bool visited[]){
    if(P == NULL) return;

    cout << P->info << " ";
    visited[P->info - 'A'] = true;

    vector<char> neighbours;
    adrEdge E = P->firstEdge;

    while(E != NULL){
        neighbours.push_back(E->node->info);
        E = E->next;
    }

    sort(neighbours.begin(), neighbours.end());

    for(char c : neighbours){
        adrNode next = FindNode(G, c);
        if(!visited[c - 'A']){
            DFS(G, next, visited);
        }
    }
}

void PrintDFS(Graph G, char start){
    adrNode P = FindNode(G, start);
    if(P == NULL){
        cout << "Node tidak ditemukan.\n";
        return;
    }

    bool visited[26] = {false};

    cout << "DFS mulai dari " << start << " : ";
    DFS(G, P, visited);
    cout << endl;
}
```


main.cpp
Modul14 > UNGUIDED > No2 > main.cpp > main()
Tabnine | Edit | Test | Explain | Document

```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 NodeG = 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, NodeG);
    ConnectNode(G, D, H);
    ConnectNode(G, E, H);
    ConnectNode(G, F, H);
    ConnectNode(G, NodeG, H);

    PrintInfoGraph(G);
    PrintDFS(G, 'A');

    return 0;
}
```

Screenshots Output

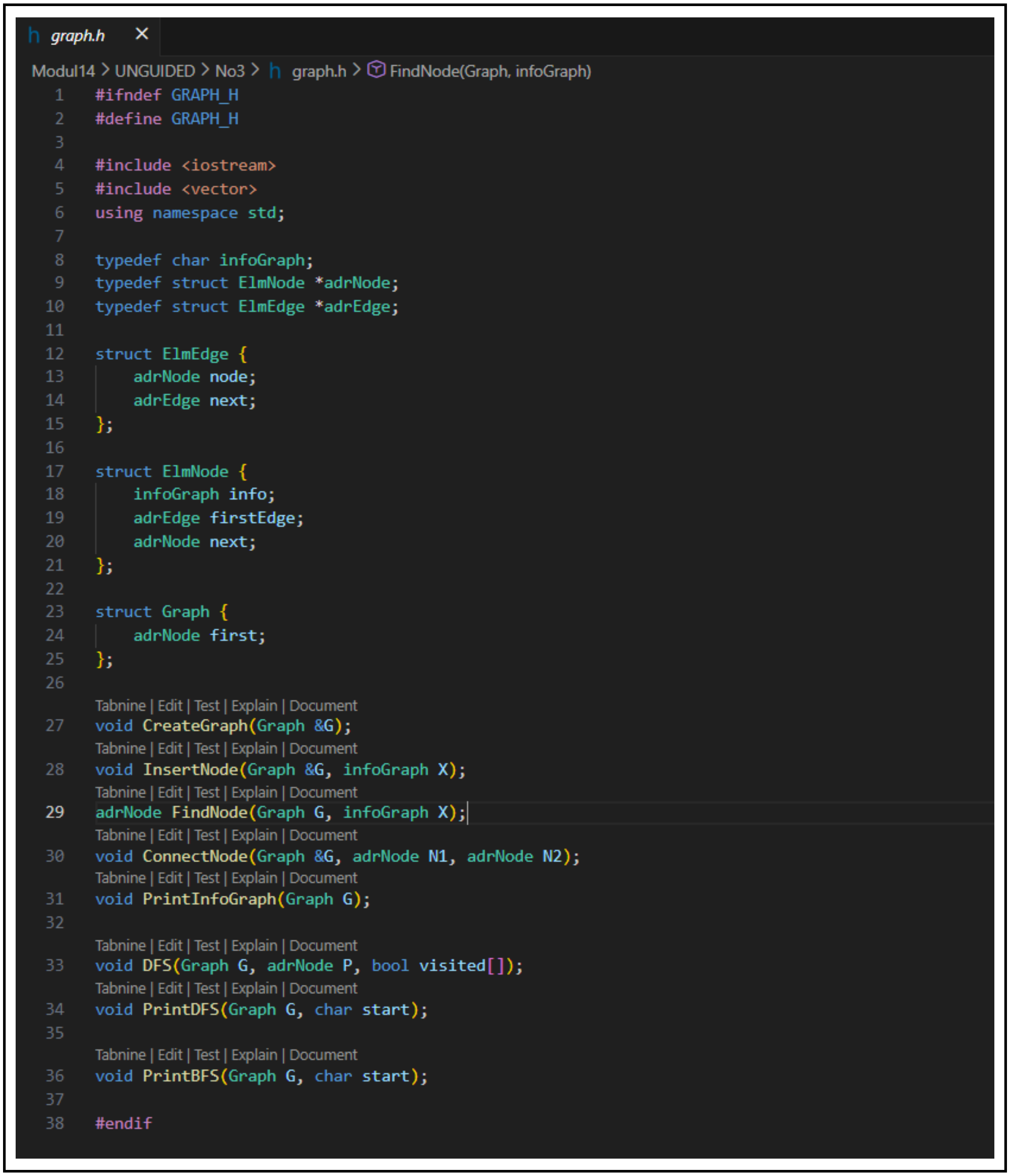


```
PS D:\StrukturData> cd "d:\StrukturData\Modul14\UNGUIDED\No2\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
A : C B
B : E D
C : G F
D : H
E : H
F : H
G : H
H :
DFS mulai dari A : A B D H E C F G
```

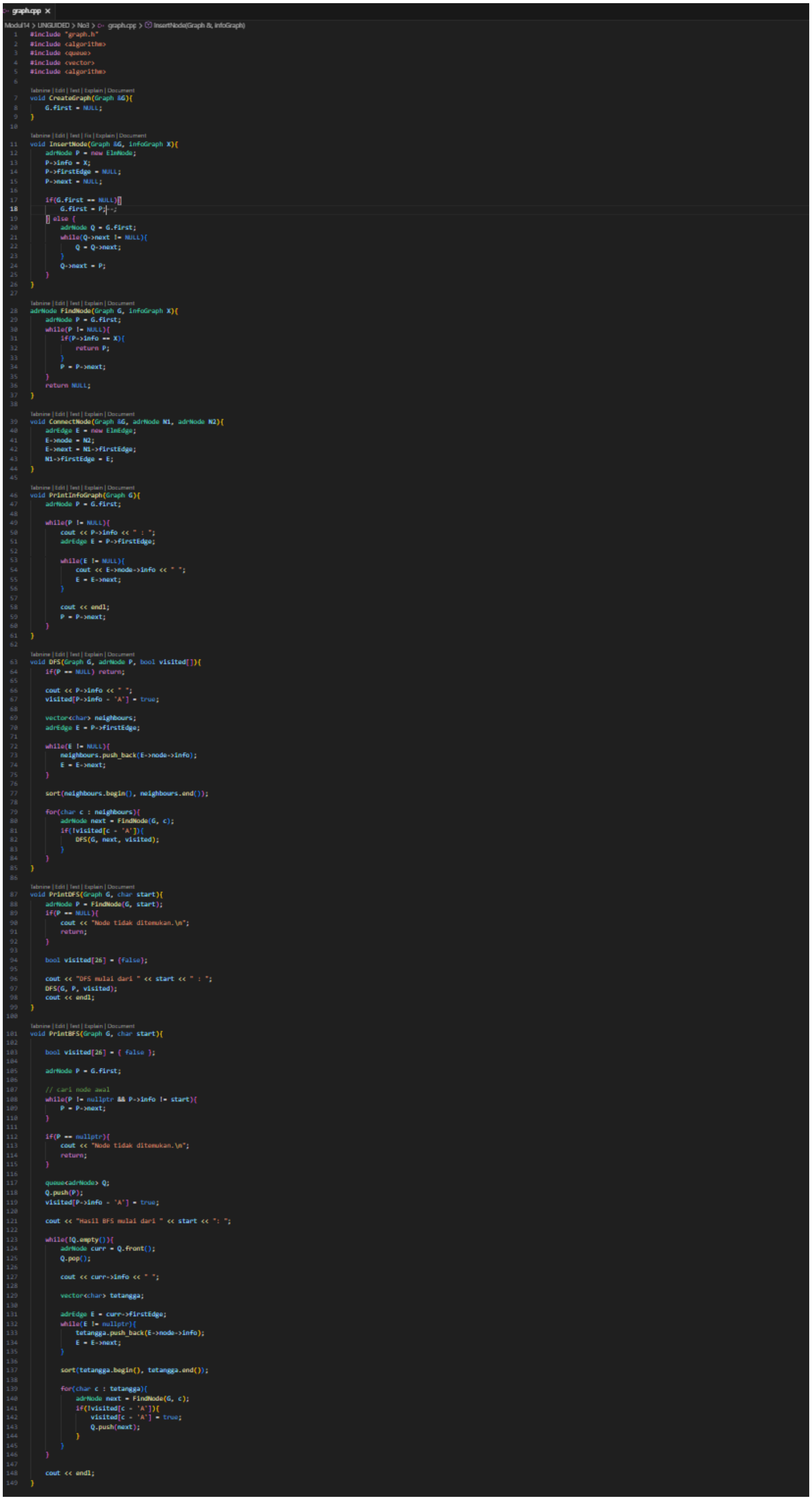
Deskripsi:

Program ini mengimplementasikan struktur data graph berarah menggunakan representasi adjacency list dalam bahasa C++. File graph.h berisi pendefinisian struktur graph, node, dan edge serta deklarasi fungsi-fungsi dasar seperti pembuatan graph, penambahan node, pencarian node, dan penelusuran graph. File graph.cpp mengimplementasikan operasi tersebut, termasuk pembuatan graph, penambahan node secara berurutan, penghubungan antar node dengan edge berarah, serta penelusuran Depth First Search (DFS). Algoritma DFS diimplementasikan secara rekursif dengan penanda kunjungan dan menggunakan pengurutan tetangga agar urutan penelusuran bersifat alfabetis. Pada file main.cpp, graph dibangun dengan beberapa node dari A hingga H dan dihubungkan sesuai struktur yang ditentukan, kemudian program menampilkan daftar ketetanggaan graph dan hasil penelusuran DFS yang dimulai dari node A.

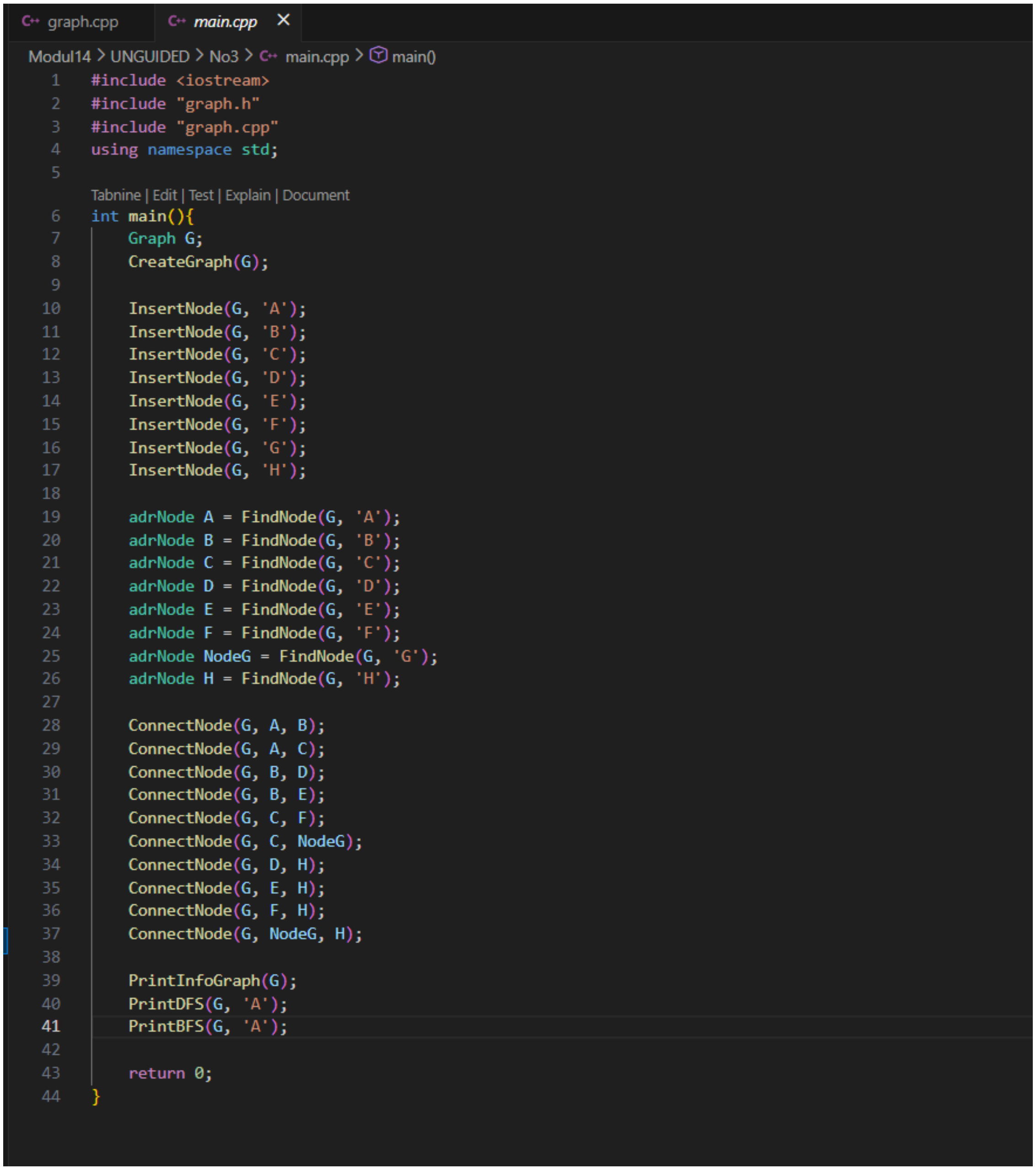
Unguided 3


graph.h
Modul14 > UNGUIDED > No3 > graph.h > FindNode(Graph, infoGraph)
Tabnine | Edit | Test | Explain | Document

```cpp
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
#include <vector>
using namespace std;

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

struct ElmEdge {
    adrNode node;
    adrEdge next;
};

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

struct Graph {
    adrNode first;
};

void CreateGraph(Graph &G);
void InsertNode(Graph &G, infoGraph X);
adrNode FindNode(Graph G, infoGraph X);
void ConnectNode(Graph &G, adrNode N1, adrNode N2);
void PrintInfoGraph(Graph G);

void DFS(Graph G, adrNode P, bool visited[]);
void PrintDFS(Graph G, char start);

void PrintBFS(Graph G, char start);

#endif
```

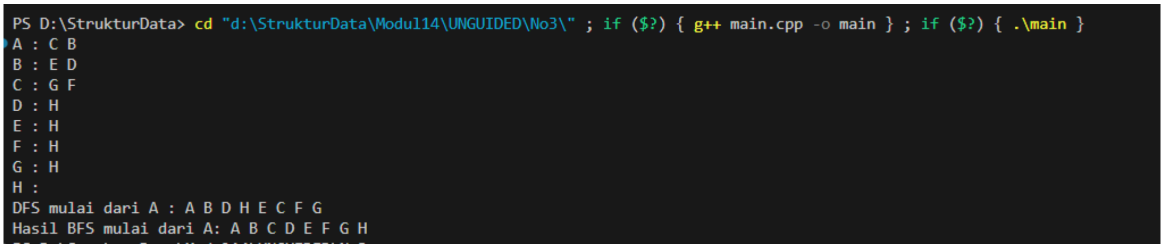

graph.cpp ×
Modul14 > UNGUIDED > No3 > graph.cpp > InsertNode(Graph &, infoGraph)
Tabnine | Edit | Test | Explain | Document
Tabnine | Edit | Test | Fix | Explain | Document

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

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

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

    if(G.first == NULL){
        G.first = P;
    } else {
        adrNode Q = G.first;
        while(Q->next != NULL){
            Q = Q->next;
        }
        Q->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 ConnectNode(Graph &G, adrNode N1, adrNode N2){
    adrEdge E = new ElmEdge;
    E->node = N2;
    E->next = N1->firstEdge;
    N1->firstEdge = E;
}

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 DFS(Graph G, adrNode P, bool visited[]){
    if(P == NULL) return;

    cout << P->info << " ";
    visited[P->info - 'A'] = true;

    vector<char> neighbours;
    adrEdge E = P->firstEdge;

    while(E != NULL){
        neighbours.push_back(E->node->info);
        E = E->next;
    }

    sort(neighbours.begin(), neighbours.end());

    for(char c : neighbours){
        adrNode next = FindNode(G, c);
        if(!visited[c - 'A']){
            DFS(G, next, visited);
        }
    }
}

void PrintDFS(Graph G, char start){
    adrNode P = FindNode(G, start);
    if(P == NULL){
        cout << "Node tidak ditemukan.\n";
        return;
    }

    bool visited[26] = {false};

    cout << "DFS mulai dari " << start << " : ";
    DFS(G, P, visited);
    cout << endl;
}

void PrintBFS(Graph G, char start){

    bool visited[26] = { false };

    adrNode P = G.first;

    // cari node awal
    while(P != nullptr && P->info != start){
        P = P->next;
    }

    if(P == nullptr){
        cout << "Node tidak ditemukan.\n";
        return;
    }

    queue<adrNode> Q;
    Q.push(P);
    visited[P->info - 'A'] = true;

    cout << "Hasil BFS mulai dari " << start << ": ";

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

        cout << curr->info << " ";

        vector<char> tetangga;

        adrEdge E = curr->firstEdge;
        while(E != nullptr){
            tetangga.push_back(E->node->info);
            E = E->next;
        }

        sort(tetangga.begin(), tetangga.end());

        for(char c : tetangga){
            adrNode next = FindNode(G, c);
            if(!visited[c - 'A']){
                visited[c - 'A'] = true;
                Q.push(next);
            }
        }
    }

    cout << endl;
}
```


graph.cpp
main.cpp
Modul14 > UNGUIDED > No3 > main.cpp > main()
Tabnine | Edit | Test | Explain | Document

```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 NodeG = 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, NodeG);
    ConnectNode(G, D, H);
    ConnectNode(G, E, H);
    ConnectNode(G, F, H);
    ConnectNode(G, NodeG, H);

    PrintInfoGraph(G);
    PrintDFS(G, 'A');
    PrintBFS(G, 'A');

    return 0;
}
```

Screenshots Output



```
PS D:\StrukturData> cd "d:\StrukturData\Modul14\UNGUIDED\No3\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
A : C B
B : E D
C : G F
D : H
E : H
F : H
G : H
H :
DFS mulai dari A : A B D H E C F G
Hasil BFS mulai dari A: A B C D E F G H
```

Deskripsi:

Program ini merealisasikan graph berarah menggunakan adjacency list yang dibangun secara dinamis dengan pointer, di mana node disimpan dalam linked list dan setiap edge ditambahkan di awal daftar ketetanggaan node asal. Ciri khas utama program ini terletak pada proses DFS dan BFS yang menghasilkan urutan traversal terkontrol, karena seluruh node tetangga dikumpulkan terlebih dahulu ke dalam struktur vector lalu diurutkan secara alfabetis sebelum ditelusuri. Pendekatan ini memastikan hasil penelusuran selalu konsisten meskipun urutan penyambungan edge berbeda. DFS diimplementasikan secara rekursif dengan penanda kunjungan berbasis indeks karakter, sedangkan BFS memanfaatkan struktur data queue untuk menelusuri graph secara bertahap dari node awal. Selain itu, program mampu menampilkan struktur graph dalam bentuk daftar ketetanggaan sehingga hubungan antar node dapat diamati dengan jelas.

D. Kesimpulan

Program yang dibuat berhasil mengimplementasikan struktur data graph berarah menggunakan representasi adjacency list secara dinamis. Melalui program ini, proses penambahan node, penghubungan antar node, serta penampilan struktur graph dapat dilakukan dengan baik. Selain itu, algoritma Depth First Search (DFS) dan Breadth First Search (BFS) berhasil diterapkan dengan urutan penelusuran yang terkontrol karena tetangga node diurutkan secara alfabetis sebelum diproses. Dengan demikian, program ini membuktikan pemahaman terhadap konsep graph dan traversal serta penerapannya dalam bahasa C++.

E. Referensi

Modul 14 Struktur Data – Graph. Program Studi Teknik Informatika.

https://www.geeksforgeeks.org/dsa/introduction-to-graphs-data-structure-and-algorithm-tutorials/

https://www.programiz.com/dsa/graph