LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XII
GRAPH

Disusun Oleh :
NAMA : ZAKI MAULA DHIA
NIM : 103112400127

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

GRAPH adalah struktur data yang memiliki kumpulan node yang terhubung oleh edge yang menggambarkan hubungan antara node-node tersebut. Algoritma GRAPH adalah serangkaian langkah atau prosedur yang dirancang untuk memanipulasi atau menganalisis data yang terstruktur dalam bentu graph. Algoritma GRAPH memungkinkan untuk melakukan berbagai operasi dalam GRAPH, termasuk analisis jaringan,optimisasi rute, pengenalan pola, dan banyak lagi.

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

Guided 1

```
// graf.h
#ifndef GRAF_H_INCLUDE
#define GRAF_H_INCLUDE

#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 PrintDFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

```
// graf.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;

    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){
```

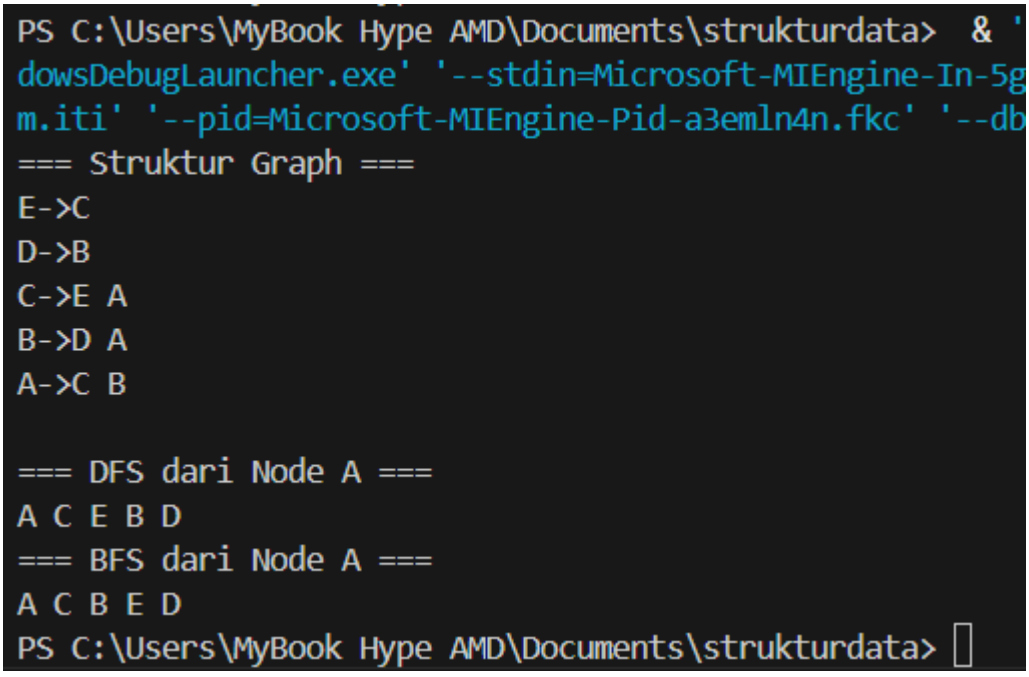
```
            cout << E->node->info << " ";
            E = E->next;
        }
        cout << endl;
        P = P->next;
    }
}

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);

    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



Deskripsi:

Program ini adalah implementasi dari struktur data GRAPH. Program tersebut menggunakan operasi dari algoritma GRAPH yang wajib digunakan, program itu juga menggunakan alogritma search DFS (Depth First Search) cara kerjanya adalah dengan menelusuri satu cabang terlebih dahulu. Program tersebut juga menggunakan algoritma search BFS (Breadth First Search) dengan cara kerja berbeda dari DFS dimana BFS menjelajahi dari root ke setiap level kedalaman terlebih dahulu kemudian ke kedalaman yang selanjutnya.

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

Unguided 1

```cpp
// main.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 << endl;
    return 0;
}
```

```cpp
// graf.h
#ifndef GRAF_H_INCLUDE
#define GRAF_H_INCLUDE

#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);

#endif
```

```
// graf.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;

    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;
    }
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata>  &
dowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-p
y.did' '--pid=Microsoft-MIEngine-Pid-zk3cpwz0.v5m' '--d
=== Struktur Graph ===
E->C
D->B
C->E A
B->D A
A->C B

PS C:\Users\MyBook Hype AMD\Documents\strukturdata>
```

Deskripsi:

Program tersebut adalah struktur data GRAPH yang akan menampilkan struktur graphnya saja, tanpa DFS ataupun BFS dikarenakan program tersebut tidak memiliki function untuk menampilkan hasil DSF ataupun BFS. Cara kerjanya sama seperti guided 1 dimana setiap node memiliki edge nya sendiri, sehingga setiap node dapat berkomunikasi.

Unguided 2

```cpp
// main.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 << endl;
    return 0;
}
```

```
// graf.h
#ifndef GRAF_H_INCLUDE
#define GRAF_H_INCLUDE

#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);
```

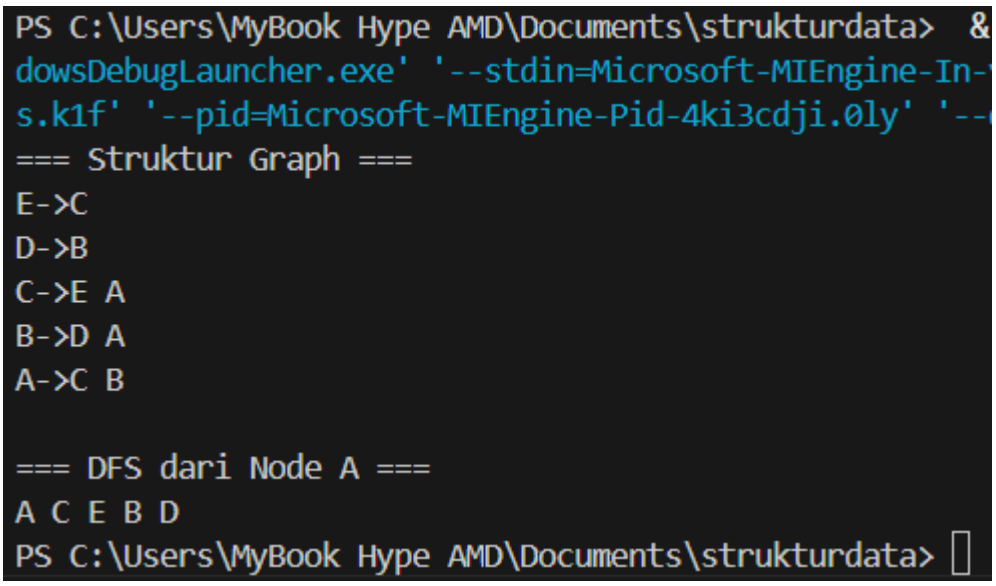
```
void ResetVisited(Graph &G);
void PrintDFS(Graph &G, adrNode N);

#endif
```

```
// graf.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;

    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;
    }
}
```

Screenshots Output



Deskripsi:

Program tersebut lanjutan dari unguided 1 yang dimana hanya ditambah function PrintDFS, dimana DFS (Depth First Search) digunakan untuk mencari sebuah node yang akan dicari dengan metode memasuki satu cabang terlebih dahulu sampai kedaun dan kemudian pindah ke cabang lain.

Unguided 3

```cpp
// main.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;
}
```

```cpp
// graf.h
```

```h
#ifndef GRAF_H_INCLUDE
#define GRAF_H_INCLUDE

#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 PrintDFS(Graph &G, adrNode N);
void PrintBFS(Graph &G, adrNode N);

#endif
```

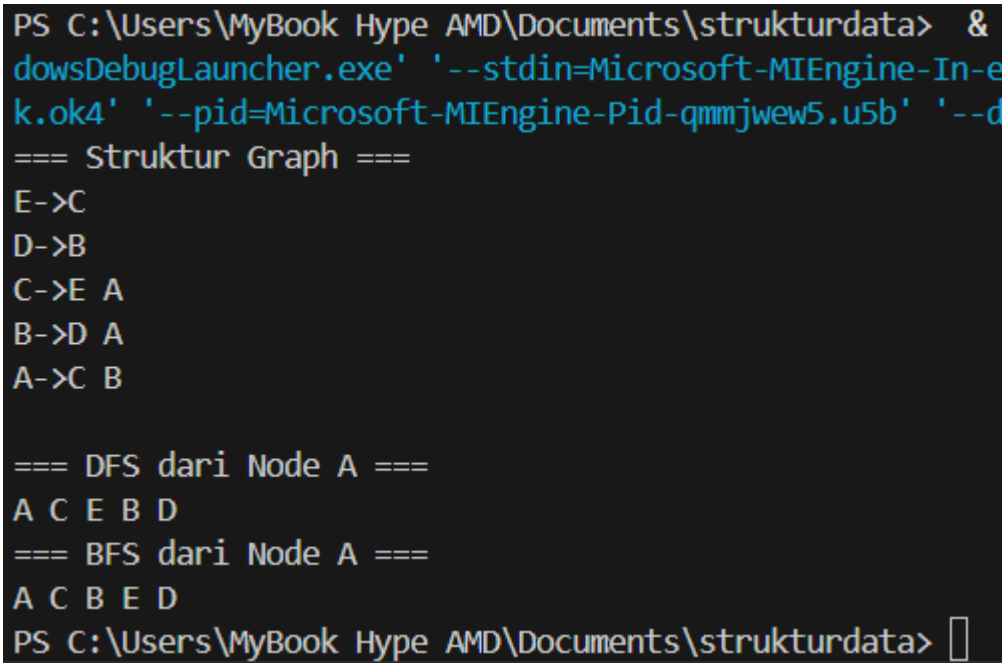
```
// graf.cpp
#include "graf.h"
#include <queue>
#include <stack>

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;

    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;
            }
        }
    }
```

```
}
```

Screenshots Output



Deskripsi:

Program tersebut lanjutan dari unguided 2 yang menambahkan function BFS (Breadth First Search) digunakan untuk mencari node yang diinginkan dengan cara membaca setiap level kedalaman dalam graph tersebut.

D. Kesimpulan

Materi GRAPH adalah struktur data yang setiap node memiliki sebuah edge sehingga setiap node terhubung pada node lain, struktur data GRAPH dapat digunakan untuk pemodelan koneksi antara node/device, seperti jaringan komputer, google maps, dll. Karena struktur data GRAPH masih lebih simpel daripada struktur data MULTI LINKED LIST, oleh sebab itu struktur data GRAPH lebih sering digunakan, bahkan setiap server yang ingin mevisualisasikan hubungan setiap node akan menggunakan struktur data GRAPH, sehingga visual dapat terlihat dan dipahami oleh manusia.

E. Referensi

- Ginting, S. H. N., Effendi, H., Kumar, S., Marsisno, W., Sitanggang, Y. R. U., Anwar, K., ... & Smrti, N. N. E. (2024). Pengantar struktur data. Penerbit Mifandi Mandiri Digital, 1(01).
- Ginting, S. H. N., Effendi, H., Kumar, S., Marsisno, W., Sitanggang, Y. R. U., Anwar, K., ... & Smrti, N. N. E. (2024). Pengantar struktur data. *Penerbit Mifandi Mandiri Digital*, *1*(01).