

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL XIV
GRAPH**



Disusun Oleh :

NAMA : Azzahra Fareluka Esti Ning Tyas

NIM : 103112430023

Dosen

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Graph adalah 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

h graf.h X

Modul14 > GUIDED > h graf.h > ...

```
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 void CreateGraph(Graph &G);
35 adrNode AllocateNode(infoGraph X);
36 adrEdge AllocateEdge(adrNode N);
37
38 void InsertNode(Graph &G, infoGraph X);
39 adrNode FindNode(Graph G, infoGraph X);
40
41 void ConnectNode(Graph &G, infoGraph A, infoGraph B);
42
43 void PrintInfoGraph(Graph G);
44
45 void ResetVisited(Graph &G);
46 void DFS(Graph &G, adrNode N);
47 void PrintBFS(Graph &G, adrNode N);
48
49 #endif
```

```

1 #include "graf.h"
2 #include <queue>
3 #include <stack>
4
5 //fungsi [Edit] Test [Implementasi] Document
6 void CreateGraph(Graph &G)
7 {
8     G.first = NULL;
9 }
10
11 //fungsi [Edit] Test [Implementasi] Document
12 adrNode AllocatedNode(InfoGraph X)
13 {
14     adrNode P = new Elnode;
15     P->info = X;
16     P->visited = 0;
17     P->firstEdge = NULL;
18     P->next = NULL;
19     return P;
20 }
21
22 //fungsi [Edit] Test [Implementasi] Document
23 adrEdge AllocatedEdge(adrNode N)
24 {
25     adrEdge P = new ElnEdge;
26     P->node = N;
27     P->next = NULL;
28     return P;
29 }
30
31 //fungsi [Edit] Test [Implementasi] Document
32 void InsertNode (Graph &G, InfoGraph X)
33 {
34     adrNode P = AllocatedNode(X);
35     P->next = G.first;
36     G.first = P;
37 }
38
39 //fungsi [Edit] Test [Implementasi] Document
40 adrNode FindNode(Graph G, InfoGraph X)
41 {
42     adrNode P = G.first;
43     while (P != NULL)
44     {
45         if(P->info == X)
46             return P;
47         P = P->next;
48     }
49     return NULL;
50 }
51
52 //fungsi [Edit] Test [Implementasi] Document
53 void ConnectNode(Graph &G, InfoGraph A, InfoGraph B)
54 {
55     adrNode N1 = FindNode(G, A);
56     adrNode N2 = FindNode(G, B);
57     if (N1 == NULL || N2 == NULL)
58     {
59         cout << "Node tidak ditemukan";
60         return;
61     }
62     //buat edge dari N1 ke N2
63     adrEdge E1 = AllocatedEdge(N1);
64     E1->next = N2->firstEdge;
65     N1->firstEdge = E1;
66     //karena undirected > buat edge balik
67     adrEdge E2 = AllocatedEdge(N2);
68     E2->next = N1->firstEdge;
69     N2->firstEdge = E2;
70 }
71
72 //fungsi [Edit] Test [Implementasi] Document
73 void PrintInfoGraph(Graph G)
74 {
75     adrNode P = G.first;
76     while (P != NULL)
77     {
78         cout << P->info << " ";
79         adrEdge E = P->firstEdge;
80         while (E != NULL)
81         {
82             cout << E->node->info << " ";
83             E = E->next;
84         }
85         cout << endl;
86         P = P->next;
87     }
88 }
89
90 //fungsi [Edit] Test [Implementasi] Document
91 void ResetVisited(Graph &G)
92 {
93     adrNode P = G.first;
94     while (P != NULL)
95     {
96         P->visited = 0;
97         P = P->next;
98     }
99 }
100
101 //fungsi [Edit] Test [Implementasi] Document
102 void PrintDFS(Graph &G, adrNode N)
103 {
104     if (N == NULL)
105         return;
106     N->visited = 1;
107     cout << N->info << " ";
108     adrEdge E = N->firstEdge;
109     while (E != NULL)
110     {
111         if (E->node->visited == 0)
112         {
113             PrintDFS(G, E->node);
114         }
115         E = E->next;
116     }
117 }
118
119 //fungsi [Edit] Test [Implementasi] Document
120 void PrintBFS(Graph &G, adrNode N)
121 {
122     if(N == NULL)
123         return;
124     queue<adrNode> Q;
125     Q.push(N);
126     while (!Q.empty())
127     {
128         adrNode curv = Q.front();
129         Q.pop();
130         if (curv->visited == 0)
131         {
132             curv->visited = 1;
133             cout << curv->info << " ";
134             adrEdge E = curv->firstEdge;
135             while (E != NULL)
136             {
137                 if (E->node->visited == 0)
138                 {
139                     Q.push(E->node);
140                 }
141                 E = E->next;
142             }
143         }
144     }
145 }

```

C++ main.cpp X

Modul14 > GUIDED > C++ main.cpp > ...

```
1  #include "graf.h"
2  #include "graf.cpp"
3  #include <iostream>
4  using namespace std;
5
   Tabnine | Edit | Test | Explain | Document
6  int main()
7  {
8      Graph G;
9      CreateGraph(G);
10
11     InsertNode(G, 'A');
12     InsertNode(G, 'B');
13     InsertNode(G, 'C');
14     InsertNode(G, 'D');
15     InsertNode(G, 'E');
16
17     ConnectNode(G, 'A', 'B');
18     ConnectNode(G, 'A', 'C');
19     ConnectNode(G, 'B', 'D');
20     ConnectNode(G, 'C', 'E');
21
22     cout << "=== Struktur Graph ===\n";
23     PrintInfoGraph(G);
24
25     cout << "\n=== DFS dari Node A ===\n";
26     ResetVisited(G);
27     PrintDFS(G, FindNode(G, 'A'));
28
29     cout << "\n=== BFS dari Node A ===\n";
30     ResetVisited(G);
31     PrintBFS(G, FindNode(G, 'A'));
32
33     cout << endl;
34     return 0;
35 }
```

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

h graph.h X

Modul14 > UNGUIDED > No1 > h graph.h > ...

```
1  #ifndef GRAPH_H
2  #define GRAPH_H
3  #include <iostream>
4  using namespace std;
5
6  typedef char infoGraph;
7  typedef struct ElmNode *adrNode;
8  typedef struct ElmEdge *adrEdge;
9
10 struct ElmEdge {
11     adrNode node;
12     adrEdge next;
13 };
14
15 struct ElmNode {
16     infoGraph info;
17     int visited;
18     adrEdge firstEdge;
19     adrNode next;
20 };
21
22 struct Graph {
23     adrNode first;
24 };
25
26 void CreateGraph(Graph &G);
27 void InsertNode(Graph &G, infoGraph X);
28 void ConnectNode(Graph &G, adrNode N1, adrNode N2);
29 void PrintInfoGraph(Graph G);
30 adrNode FindNode(Graph G, infoGraph X);
31
32 #endif
```

C++ graph.cpp X

Modul14 > UNGUIDED > No1 > C++ graph.cpp > PrintInfoGraph(Graph)

```
1  #include "graph.h"
2
3  Tabnine | Edit | Test | Explain | Document
4  void CreateGraph(Graph &G){
5      G.first = NULL;
6  }
7
8  Tabnine | Edit | Test | Explain | Document
9  void InsertNode(Graph &G, infoGraph X){
10     adrNode P = new ElmNode;
11     P->info = X;
12     P->visited = 0;
13     P->firstEdge = NULL;
14     P->next = NULL;
15
16     if(G.first == NULL){
17         G.first = P;
18     } else {
19         adrNode Q = G.first;
20         while(Q->next != NULL){
21             Q = Q->next;
22         }
23         Q->next = P;
24     }
25 }
26
27 Tabnine | Edit | Test | Explain | Document
28 adrNode FindNode(Graph G, infoGraph X){
29     adrNode P = G.first;
30     while(P != NULL){
31         if(P->info == X){
32             return P;
33         }
34         P = P->next;
35     }
36     return NULL;
37 }
38
39 Tabnine | Edit | Test | Explain | Document
40 void ConnectNode(Graph &G, adrNode N1, adrNode N2){
41     if(N1 != NULL && N2 != NULL){
42         adrEdge E = new ElmEdge;
43         E->node = N2;
44         E->next = N1->firstEdge;
45         N1->firstEdge = E;
46     }
47 }
48
49 Tabnine | Edit | Test | Explain | Document
50 void PrintInfoGraph(Graph G){
51     adrNode P = G.first;
52     while(P != NULL){
53         cout << "Node " << P->info << " terhubung ke: ";
54
55         adrEdge E = P->firstEdge;
56         while(E != NULL){
57             cout << E->node->info << " ";
58             E = E->next;
59         }
60
61         cout << endl;
62         P = P->next;
63     }
64 }
```



```
main.cpp X
Modul14 > UNGUIDED > No1 > C++ main.cpp > ...
1  #include <iostream>
2  #include "graph.h"
3  #include "graph.cpp"
4  using namespace std;
5
6  Tabnine | Edit | Test | Explain | Document
7  int main(){
8      Graph G;
9      CreateGraph(G);
10
11     InsertNode(G, 'A');
12     InsertNode(G, 'B');
13     InsertNode(G, 'C');
14     InsertNode(G, 'D');
15     InsertNode(G, 'E');
16     InsertNode(G, 'F');
17     InsertNode(G, 'G');
18     InsertNode(G, 'H');
19
20     adrNode A = FindNode(G, 'A');
21     adrNode B = FindNode(G, 'B');
22     adrNode C = FindNode(G, 'C');
23     adrNode D = FindNode(G, 'D');
24     adrNode E = FindNode(G, 'E');
25     adrNode F = FindNode(G, 'F');
26     adrNode NodeG = FindNode(G, 'G');
27     adrNode H = FindNode(G, 'H');
28
29     ConnectNode(G, A, B);
30     ConnectNode(G, A, C);
31     ConnectNode(G, B, D);
32     ConnectNode(G, B, E);
33     ConnectNode(G, C, F);
34     ConnectNode(G, C, NodeG);
35     ConnectNode(G, D, H);
36     ConnectNode(G, E, H);
37     ConnectNode(G, F, H);
38     ConnectNode(G, NodeG, H);
39
40     PrintInfoGraph(G);
41
42     return 0;
43 }
```

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

h graph.h x

Modul14 > UNGUIDED > No2 > h graph.h > ...

```
1  #ifndef GRAPH_H
2  #define GRAPH_H
3
4  #include <iostream>
5  #include <vector>
6  using namespace std;
7
8  typedef char infoGraph;
9  typedef struct ElmNode *adrNode;
10 typedef struct ElmEdge *adrEdge;
11
12 struct ElmEdge {
13     adrNode node;
14     adrEdge next;
15 };
16
17 struct ElmNode {
18     infoGraph info;
19     adrEdge firstEdge;
20     adrNode next;
21 };
22
23 struct Graph {
24     adrNode first;
25 };
26
27 void CreateGraph(Graph &G);
28 void InsertNode(Graph &G, infoGraph X);
29 adrNode FindNode(Graph G, infoGraph X);
30 void ConnectNode(Graph &G, adrNode N1, adrNode N2);
31 void PrintInfoGraph(Graph G);
32
33 void DFS(Graph G, adrNode P, bool visited[]);
34 void PrintDFS(Graph G, char start);
35
36 #endif
```

C++ graph.cpp X
Modul14 > UNGUIDED > No2 > C++ graph.cpp > InsertNode(Graph &, infoGraph)

```
1 #include "graph.h"
2 #include <algorithm>
3
4 Tabnine | Edit | Test | Explain | Document
5 void CreateGraph(Graph &G){
6     G.first = NULL;
7 }
8
9 Tabnine | Edit | Test | Explain | Document
10 void InsertNode(Graph &G, infoGraph X){
11     adrNode P = new ElmNode;
12     P->info = X;
13     P->firstEdge = NULL;
14     P->next = NULL;
15
16     if(G.first == NULL){
17         G.first = P;
18     } else {
19         adrNode Q = G.first;
20         while(Q->next != NULL){
21             Q = Q->next;
22         }
23         Q->next = P;
24     }
25 }
26
27 Tabnine | Edit | Test | Explain | Document
28 adrNode FindNode(Graph G, infoGraph X){
29     adrNode P = G.first;
30     while(P != NULL){
31         if(P->info == X){
32             return P;
33         }
34         P = P->next;
35     }
36     return NULL;
37 }
38
39 Tabnine | Edit | Test | Explain | Document
40 void ConnectNode(Graph &G, adrNode N1, adrNode N2){
41     adrEdge E = new ElmEdge;
42     E->node = N2;
43     E->next = N1->firstEdge;
44     N1->firstEdge = E;
45 }
46
47 Tabnine | Edit | Test | Explain | Document
48 void PrintInfoGraph(Graph G){
49     adrNode P = G.first;
50
51     while(P != NULL){
52         cout << P->info << " : ";
53         adrEdge E = P->firstEdge;
54
55         while(E != NULL){
56             cout << E->node->info << " ";
57             E = E->next;
58         }
59
60         cout << endl;
61         P = P->next;
62     }
63 }
64
65 Tabnine | Edit | Test | Explain | Document
66 void DFS(Graph G, adrNode P, bool visited[]){
67     if(P == NULL) return;
68
69     cout << P->info << " ";
70     visited[P->info - 'A'] = true;
71
72     vector<char> neighbours;
73     adrEdge E = P->firstEdge;
74
75     while(E != NULL){
76         neighbours.push_back(E->node->info);
77         E = E->next;
78     }
79
80     sort(neighbours.begin(), neighbours.end());
81
82     for(char c : neighbours){
83         adrNode next = FindNode(G, c);
84         if(!visited[c - 'A']){
85             DFS(G, next, visited);
86         }
87     }
88 }
89
90 Tabnine | Edit | Test | Explain | Document
91 void PrintDFS(Graph G, char start){
92     adrNode P = FindNode(G, start);
93     if(P == NULL){
94         cout << "Node tidak ditemukan.\n";
95         return;
96     }
97
98     bool visited[26] = {false};
99
100     cout << "DFS mulai dari " << start << " : ";
101     DFS(G, P, visited);
102     cout << endl;
103 }
```

```
C++ main.cpp X
Modul14 > UNGUIDED > No2 > C++ main.cpp > main()
1  #include <iostream>
2  #include "graph.h"
3  #include "graph.cpp"
4  using namespace std;
5
6  Tabnine | Edit | Test | Explain | Document
7  int main(){
8      Graph G;
9      CreateGraph(G);
10
11     InsertNode(G, 'A');
12     InsertNode(G, 'B');
13     InsertNode(G, 'C');
14     InsertNode(G, 'D');
15     InsertNode(G, 'E');
16     InsertNode(G, 'F');
17     InsertNode(G, 'G');
18     InsertNode(G, 'H');
19
20     adrNode A = FindNode(G, 'A');
21     adrNode B = FindNode(G, 'B');
22     adrNode C = FindNode(G, 'C');
23     adrNode D = FindNode(G, 'D');
24     adrNode E = FindNode(G, 'E');
25     adrNode F = FindNode(G, 'F');
26     adrNode NodeG = FindNode(G, 'G');
27     adrNode H = FindNode(G, 'H');
28
29     ConnectNode(G, A, B);
30     ConnectNode(G, A, C);
31     ConnectNode(G, B, D);
32     ConnectNode(G, B, E);
33     ConnectNode(G, C, F);
34     ConnectNode(G, C, NodeG);
35     ConnectNode(G, D, H);
36     ConnectNode(G, E, H);
37     ConnectNode(G, F, H);
38     ConnectNode(G, NodeG, H);
39
40     PrintInfoGraph(G);
41     PrintDFS(G, 'A');
42
43     return 0;
44 }
```

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

h graph.h X

Modul14 > UNGUIDED > No3 > h graph.h > FindNode(Graph, infoGraph)

```
1  #ifndef GRAPH_H
2  #define GRAPH_H
3
4  #include <iostream>
5  #include <vector>
6  using namespace std;
7
8  typedef char infoGraph;
9  typedef struct ElmNode *adrNode;
10 typedef struct ElmEdge *adrEdge;
11
12 struct ElmEdge {
13     adrNode node;
14     adrEdge next;
15 };
16
17 struct ElmNode {
18     infoGraph info;
19     adrEdge firstEdge;
20     adrNode next;
21 };
22
23 struct Graph {
24     adrNode first;
25 };
26
27 void CreateGraph(Graph &G);
28 void InsertNode(Graph &G, infoGraph X);
29 adrNode FindNode(Graph G, infoGraph X);
30 void ConnectNode(Graph &G, adrNode N1, adrNode N2);
31 void PrintInfoGraph(Graph G);
32
33 void DFS(Graph G, adrNode P, bool visited[]);
34 void PrintDFS(Graph G, char start);
35
36 void PrintBFS(Graph G, char start);
37
38 #endif
```

```

graph TD
    subgraph "Module 4 UNRELEASED"
        direction TB
        subgraph "Main"
            direction TB
            N1["Noi"]
            N2["G"]
            N3["graphviz"]
            N4["infograph"]
        end
        subgraph "Graphs"
            direction TB
            G1["Graph G"]
            G2["Graph H"]
            G3["Graph I"]
            G4["Graph J"]
        end
        subgraph "Algorithms"
            direction TB
            A1["Algorithm 1"]
            A2["Algorithm 2"]
            A3["Algorithm 3"]
            A4["Algorithm 4"]
        end
        subgraph "Data Structures"
            direction TB
            DS1["Data Structure 1"]
            DS2["Data Structure 2"]
            DS3["Data Structure 3"]
            DS4["Data Structure 4"]
        end
        subgraph "Utilities"
            direction TB
            U1["Utility 1"]
            U2["Utility 2"]
            U3["Utility 3"]
            U4["Utility 4"]
        end
    end

```



```
C++ graph.cpp  C++ main.cpp X
Modul14 > UNGUIDED > No3 > C++ main.cpp > main()
1  #include <iostream>
2  #include "graph.h"
3  #include "graph.cpp"
4  using namespace std;
5
6  int main(){
7      Graph G;
8      CreateGraph(G);
9
10     InsertNode(G, 'A');
11     InsertNode(G, 'B');
12     InsertNode(G, 'C');
13     InsertNode(G, 'D');
14     InsertNode(G, 'E');
15     InsertNode(G, 'F');
16     InsertNode(G, 'G');
17     InsertNode(G, 'H');
18
19     adrNode A = FindNode(G, 'A');
20     adrNode B = FindNode(G, 'B');
21     adrNode C = FindNode(G, 'C');
22     adrNode D = FindNode(G, 'D');
23     adrNode E = FindNode(G, 'E');
24     adrNode F = FindNode(G, 'F');
25     adrNode NodeG = FindNode(G, 'G');
26     adrNode H = FindNode(G, 'H');
27
28     ConnectNode(G, A, B);
29     ConnectNode(G, A, C);
30     ConnectNode(G, B, D);
31     ConnectNode(G, B, E);
32     ConnectNode(G, C, F);
33     ConnectNode(G, C, NodeG);
34     ConnectNode(G, D, H);
35     ConnectNode(G, E, H);
36     ConnectNode(G, F, H);
37     ConnectNode(G, NodeG, H);
38
39     PrintInfoGraph(G);
40     PrintDFS(G, 'A');
41     PrintBFS(G, 'A');
42
43     return 0;
44 }
```

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>