

**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 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 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);  
Tabnine | Edit | Test | Explain | Document  
38 void InsertNode(Graph &G, infoGraph X);  
Tabnine | Edit | Test | Explain | Document  
39 adrNode FindNode(Graph G, infoGraph X);  
Tabnine | Edit | Test | Explain | Document  
40 void ConnectNode(Graph &G, infoGraph A, infoGraph B);  
Tabnine | Edit | Test | Explain | Document  
41 void PrintInfoGraph(Graph G);  
42  
43 Tabnine | Edit | Test | Explain | Document  
44 void ResetVisited(Graph &G);  
Tabnine | Edit | Test | Explain | Document  
45 void DFS(Graph &G, adrNode N);  
Tabnine | Edit | Test | Explain | Document  
46 void PrintBFS(Graph &G, adrNode N);  
47  
48  
49 #endif
```

```

  graf.cpp x
Modul(> GUIDED > <-> graf.cpp > AllocateNode(InfoGraph)
1 #include "graf.h"
2 #include <queue>
3 #include <stack>
4
5 IaJinme | Edit | Test | Explain | Document
6 void CreateGraph(Graph &G)
7 {
8     G.first = NULL;
9 }
10
11 IaJinme | Edit | Test | Explain | Document
12 adrNode AllocateNode(InfoGraph X)
13 {
14     adrNode P = new ElNode;
15     P->info = X;
16     P->next = NULL;
17     P->firstEdge = NULL;
18     P->next = NULL;
19 }
20
21 IaJinme | Edit | Test | Explain | Document
22 adrEdge AllocateEdge(adrNode N)
23 {
24     adrEdge P = new ElEdge;
25     P->node = N;
26     P->next = NULL;
27     return P;
28 }
29
30 IaJinme | Edit | Test | Explain | Document
31 void InsertNode(Graph &G, InfoGraph X)
32 {
33     adrNode P = AllocateNode(X);
34     P->next = G.first;
35     G.first = P;
36 }
37
38 IaJinme | Edit | Test | Explain | Document
39 adrNode FindNode(Graph G, InfoGraph X)
40 {
41     adrNode P = G.first;
42     while (P != NULL)
43     {
44         if (P->info == X)
45             return P;
46         P = P->next;
47     }
48     return NULL;
49 }
50
51 IaJinme | Edit | Test | Explain | Document
52 void ConnectNode(Graph &G, InfoGraph A, InfoGraph B)
53 {
54     adrNode N1 = FindNode(G, A);
55     adrNode N2 = FindNode(G, B);
56
57     if (N1 == NULL || N2 == NULL)
58     {
59         cout << "Node tidak ditemukan\n";
60         return;
61     }
62
63     //Buat edge dari N1 ke N2
64     adrEdge E1 = AllocateEdge(N2);
65     E1->next = N1->firstEdge;
66     N1->firstEdge = E1;
67
68     //Karena undirected -> buat edge balik
69     adrEdge E2 = AllocateEdge(N1);
70     E2->next = N2->firstEdge;
71     N2->firstEdge = E2;
72 }
73
74 IaJinme | Edit | Test | Explain | Document
75 void PrintGraph(Graph G)
76 {
77     adrNode P = G.first;
78     while (P != NULL)
79     {
80         cout << P->info << " -> ";
81         adrEdge E = P->firstEdge;
82         while (E != NULL)
83         {
84             cout << E->node->info << " ";
85             E = E->next;
86         }
87         cout << endl;
88         P = P->next;
89     }
90 }
91
92 IaJinme | Edit | Test | Explain | Document
93 void Resetvisited(Graph &G)
94 {
95     adrNode P = G.first;
96     while (P != NULL)
97     {
98         P->visited = 0;
99         P = P->next;
100    }
101 }
102
103 IaJinme | Edit | Test | Explain | Document
104 void PrintDFS(Graph &G, adrNode N)
105 {
106     if (N == NULL)
107         return;
108
109     N->visited = 1;
110     cout << N->info << " ";
111     adrEdge E = N->firstEdge;
112
113     while (E != NULL)
114     {
115         if (E->node->visited == 0)
116         {
117             PrintDFS(G, E->node);
118         }
119         E = E->next;
120     }
121 }
122
123 IaJinme | Edit | Test | Explain | Document
124 void PrintBFS(Graph &G, adrNode N)
125 {
126     if (N == NULL)
127         return;
128
129     queue<adrNode> Q;
130     Q.push(N);
131
132     while (!Q.empty())
133     {
134         adrNode curr = Q.front();
135         Q.pop();
136
137         if (curr->visited == 0)
138         {
139             curr->visited = 1;
140             cout << curr->info << " ";
141
142             adrEdge E = curr->firstEdge;
143             while (E != NULL)
144             {
145                 if (E->node->visited == 0)
146                 {
147                     Q.push(E->node);
148                 }
149                 E = E->next;
150             }
151         }
152     }
153 }

```

### 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
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 Tabnine | Edit | Test | Explain | Document
27 adrNode FindNode(Graph G, infoGraph X){
28    adrNode P = G.first;
29    while(P != NULL){
30        if(P->info == X){
31            return P;
32        }
33        P = P->next;
34    }
35
36 Tabnine | Edit | Test | Explain | Document
37 void ConnectNode(Graph &G, adrNode N1, adrNode N2){
38    if(N1 != NULL && N2 != NULL){
39        adrEdge E = new ElmEdge;
40        E->node = N2;
41        E->next = N1->firstEdge;
42        N1->firstEdge = E;
43    }
44
45 Tabnine | Edit | Test | Explain | Document
46 void PrintInfoGraph(Graph G){
47    adrNode P = G.first;
48    while(P != NULL){
49        cout << "Node " << P->info << " terhubung ke: ";
50
51        adrEdge E = P->firstEdge;
52        while(E != NULL){
53            cout << E->node->info << " ";
54            E = E->next;
55        }
56
57        cout << endl;
58        P = P->next;
59    }
}
```

```
C++ 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;
}
```

## 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-fungsinya. 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  X
Modul14 > UNGUIDED > No2 > 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  Tabnine | Edit | Test | Explain | Document
28  void CreateGraph(Graph &G);
29  Tabnine | Edit | Test | Explain | Document
30  void InsertNode(Graph &G, infoGraph X);
31  Tabnine | Edit | Test | Explain | Document
32  adrNode FindNode(Graph G, infoGraph X);
33  Tabnine | Edit | Test | Explain | Document
34  void ConnectNode(Graph &G, adrNode N1, adrNode N2);
35  Tabnine | Edit | Test | Explain | Document
36  void PrintInfoGraph(Graph G);
37
38  Tabnine | Edit | Test | Explain | Document
39  void DFS(Graph G, adrNode P, bool visited[]);
40  Tabnine | Edit | Test | Explain | Document
41  void PrintDFS(Graph G, char start);
42
43  #endif
```

```
C++ graph.cpp ×
Modul14 > UNGUIDED > No2 > 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 Tabnine | Edit | Test | Explain | Document
27 void FindNode(Graph G, infoGraph X){
28     adrNode P = G.first;
29     while(P != NULL){
30         if(P->info == X){
31             return P;
32         }
33         P = P->next;
34     }
35
36 Tabnine | Edit | Test | Explain | Document
37 void ConnectNode(Graph &G, adrNode N1, adrNode N2){
38     adrEdge E = new ElmEdge;
39     E->node = N2;
40     E->next = N1->firstEdge;
41     N1->firstEdge = E;
42 }
43
44 Tabnine | Edit | Test | Explain | Document
45 void PrintInfoGraph(Graph G){
46     adrNode P = G.first;
47
48     while(P != NULL){
49         cout << P->info << " : ";
50         adrEdge E = P->firstEdge;
51
52         while(E != NULL){
53             cout << E->node->info << " ";
54             E = E->next;
55         }
56
57         cout << endl;
58         P = P->next;
59     }
60
61 Tabnine | Edit | Test | Explain | Document
62 void DFS(Graph G, adrNode P, bool visited[]){
63     if(P == NULL) return;
64
65     cout << P->info << " ";
66     visited[P->info - 'A'] = true;
67
68     vector<char> neighbours;
69     adrEdge E = P->firstEdge;
70
71     while(E != NULL){
72         neighbours.push_back(E->node->info);
73         E = E->next;
74     }
75
76     sort(neighbours.begin(), neighbours.end());
77
78     for(char c : neighbours){
79         adrNode next = FindNode(G, c);
80         if(!visited[c - 'A']){
81             DFS(G, next, visited);
82         }
83     }
84
85 Tabnine | Edit | Test | Explain | Document
86 void PrintDFS(Graph G, char start){
87     adrNode P = FindNode(G, start);
88     if(P == NULL){
89         cout << "Node tidak ditemukan.\n";
90         return;
91     }
92
93     bool visited[26] = {false};
94
95     cout << "DFS mulai dari " << start << " : ";
96     DFS(G, P, visited);
97     cout << endl;
98 }
```

```
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  
Tabnine | Edit | Test | Explain | Document  
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  
42    return 0;  
43 }
```

## 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 Tabnine | Edit | Test | Explain | Document
28 void CreateGraph(Graph &G);
29 Tabnine | Edit | Test | Explain | Document
30 void InsertNode(Graph &G, infoGraph X);
31 Tabnine | Edit | Test | Explain | Document
32 adrNode FindNode(Graph G, infoGraph X);
33 Tabnine | Edit | Test | Explain | Document
34 void ConnectNode(Graph &G, adrNode N1, adrNode N2);
35 Tabnine | Edit | Test | Explain | Document
36 void PrintInfoGraph(Graph G);
37
38 void DFS(Graph G, adrNode P, bool visited[]);
39 Tabnine | Edit | Test | Explain | Document
40 void PrintDFS(Graph G, char start);
41
42 Tabnine | Edit | Test | Explain | Document
43 void PrintBFS(Graph G, char start);
44
45 #endif
```

```

graphviz X
Module4 UNLINKED > Node > graphviz > InsertNode(Graph G, InfoGraph)
1  #include "graph.h"
2  #include <algorithm>
3  #include <queue>
4  #include <vector>
5  #include <algorithm>
6
7  /*include [edit] [text] [button] [document]
8  void CreateGraph(Graph G){*/
9  |  G.first = NULL;
9  |
10 }
11 /*include [edit] [text] [button] [document]
12 void InsertNode(Graph G, InfoGraph X){
13     adrNode P = new ElmNode;
14     P->Info = X;
15     P->FirstEdge = NULL;
16     P->next = NULL;
17
18     if(G.first == NULL){
19         G.first = P;
20     } else {
21         adrNode Q = G.first;
22         while(Q->next != NULL){
23             Q = Q->next;
24         }
25         Q->next = P;
26     }
27 }
28 /*include [edit] [text] [button] [document]
29 adrNode FindNode(Graph G, InfoGraph X){
30     adrNode P = G.first;
31     while(P != NULL){
32         if(P->Info == X){
33             return P;
34         }
35         P = P->next;
36     }
37     return NULL;
38 }
39 /*include [edit] [text] [button] [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 /*include [edit] [text] [button] [document]
47 void PrintInfoGraph(Graph G){
48     adrNode P = G.first;
49
50     while(P != NULL){
51         cout << P->Info << " ";
52         adrEdge E = P->FirstEdge;
53
54         while(E != NULL){
55             cout << E->node->Info << " ";
56             E = E->next;
57         }
58         cout << endl;
59         P = P->next;
60     }
61 }
62 /*include [edit] [text] [button] [document]
63 void DFS(Graph G, adrNode P, bool visited[]){
64     if(P == NULL) return;
65
66     cout << P->Info << " ";
67     visited[P->Info - 'A'] = true;
68
69     vector<char> neighbours;
70     adrEdge E = P->FirstEdge;
71
72     while(E != NULL){
73         neighbours.push_back(E->node->Info);
74         E = E->next;
75     }
76
77     sort(neighbours.begin(), neighbours.end());
78
79     for(char c : neighbours){
80         adrNode next = FindNode(G, c);
81         if(!visited[c - 'A']){
82             DFS(G, next, visited);
83         }
84     }
85 }
86 /*include [edit] [text] [button] [document]
87 void PrintDFS(Graph G, char start){
88     adrNode P = FindNode(G, start);
89     if(P == NULL) {
90         cout << "Node tidak ditemukan.\n";
91         return;
92     }
93
94     bool visited[26] = {false};
95
96     cout << "BFS mulai dari " << start << " : ";
97     DFS(G, P, visited);
98     cout << endl;
99 }
100 /*include [edit] [text] [button] [document]
101 void PrintBFS(Graph G, char start){
102
103     bool visited[26] = { false };
104
105     adrNode P = G.first;
106
107     // case node awal
108     while(P != NULL && P->Info != start){
109         P = P->next;
110     }
111
112     if(P == NULL){
113         cout << "Node tidak ditemukan.\n";
114         return;
115     }
116
117     queue<adrNode> Q;
118     Q.push(P);
119     visited[P->Info - 'A'] = true;
120
121     cout << "Null BFS mulai dari " << start << " : ";
122
123     while(!Q.empty()){
124         adrNode curr = Q.front();
125         Q.pop();
126
127         cout << curr->Info << " ";
128
129         vector<char> tetangga;
130
131         adrEdge E = curr->FirstEdge;
132         while(E != NULL){
133             if(E->node->Info == start){
134                 tetangga.push_back(E->node->Info);
135             }
136             E = E->next;
137         }
138
139         sort(tetangga.begin(), tetangga.end());
140
141         for(char c : tetangga){
142             adrNode next = FindNode(G, c);
143             if(!visited[c - 'A']){
144                 visited[c - 'A'] = true;
145                 Q.push(next);
146             }
147         }
148     }
149     cout << endl;
150 }

```

```

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>