LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 14
GRAPH

Disusun Oleh :
Nama: Dina Nadhyfa
NIM : 103112430052

Dosen
Fahrudin Mukti Wibowo

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Struktur data graph (graf) merupakan representasi matematis dari sekumpulan simpul (vertex) dan sisi (edge) yang menghubungkan pasangan simpul untuk memodelkan relasi antar objek, misalnya jaringan komputer, jejaring sosial, atau peta jalan. Graf dapat berbentuk berarah (directed graph) maupun tak berarah (undirected graph), serta bisa berbobot (weighted) jika setiap sisi memiliki nilai seperti jarak atau biaya. Dalam konteks struktur data, graf biasanya direpresentasikan secara internal menggunakan adjacency list atau adjacency matrix, di mana adjacency list lebih efisien untuk graf jarang (sparse) karena hanya menyimpan tetangga yang benar-benar terhubung. Beberapa penelitian di Indonesia menekankan bahwa pemilihan representasi ini memengaruhi kompleksitas waktu dan memori dalam operasi penelusuran, pencarian rute terpendek, maupun pengelompokan data berbasis graf.

Pada pemrograman C++, struktur data graph umumnya diimplementasikan menggunakan kombinasi struktur dasar seperti array/vector dan linked list, misalnya adjacency list yang disusun dengan std::vector<std::vector<int>> atau daftar tetangga berbasis pointer. Implementasi ini memudahkan penerapan algoritma penelusuran seperti Breadth First Search (BFS) dan Depth First Search (DFS), di mana BFS memanfaatkan struktur data queue untuk mengunjungi node per level, sedangkan DFS menggunakan stack (eksplisit maupun rekursi) untuk penelusuran mendalam. Modul praktikum dan makalah struktur data graf di beberapa perguruan tinggi Indonesia menunjukkan bahwa C++ menyediakan fleksibilitas tinggi untuk membangun graf berarah, tak berarah, maupun berbobot, serta mengintegrasikan algoritma seperti pencarian jalur terpendek dan minimum spanning tree dengan tetap memperhatikan efisiensi memori dan waktu eksekusi.

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

Graf.h

```
#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 AllocatedNode(infoGraph X);
adrEdge AllocatedEdge(adrNode N);

void insertNode(Graph &G, infoGraph X);
void 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 AllocatedNode(infoGraph X) {
    adrNode P = new ElmNode;
    P->info = X;
    P->visited = 0;
    P->firstEdge = NULL;
    P->next = NULL;
    return P;
}
```

```
adrEdge AllocatedEdge(adrNode N) {
    adrEdge P = new ElmEdge;
    P->node = N;
    P->next = NULL;
    return P;
}

void InsertNode(Graph &G, infoGraph X) {
    adrNode P = AllocatedNode(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 = AllocatedEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    // karena undirected -> buat edge balik
    adrEdge E2 = AllocatedEdge(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);

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

    // hubungakan node (graf tidak berarah)
    ConnectNode(G, 'A', 'B');
    ConnectNode(G, 'A', 'C');
    ConnectNode(G, 'B', 'D');
    ConnectNode(G, 'C', 'E');

    cout  << "=== Struktur Graph ===" << endl;
    printInfoGraph(G);

    cout << "\n=== Traversal DFS dari node A ===" << endl;
    ResetVisited(G);
    printDFS(G, findNode(G, 'A'));

    cout << "\n\n=== Traversal BFS dari node A ===" << endl;
    ResetVisited(G);
    printBFS(G, findNode(G, 'A'));

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

Screenshots Output

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 13\Guided> cd "d:\COOLYEAH\SEMESTER 3\Strukdat\Modul 13\Guided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
=== Struktur Graph ===
E->C
D->B
C->E A
B->D A
A->C B

=== Traversal DFS dari node A ===
A C E B D

=== Traversal BFS dari node A ===
A C B E D
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 13\Guided>
```

Deskripsi:

Program di atas merupakan implementasi struktur data graph tidak berarah (undirected graph). Graph direpresentasikan dengan node (ElmNode) yang menyimpan informasi berupa karakter, penanda kunjungan (visited), pointer ke daftar edge, dan pointer ke node berikutnya. Setiap edge (ElmEdge) menunjuk ke node tujuan dan edge berikutnya. Program menyediakan fungsi untuk membuat graph, menambah node, mencari node, menghubungkan dua node dengan edge dua arah, serta menampilkan struktur graph. Selain itu, program juga mengimplementasikan dua metode traversal graph, yaitu Depth First Search (DFS) secara rekursif dan Breadth First Search (BFS) menggunakan struktur data queue. Pada fungsi main, graph diinisialisasi dengan lima node (A, B, C, D, dan E), kemudian node-node tersebut dihubungkan sehingga membentuk graph tidak berarah. Program menampilkan struktur graph dalam bentuk adjacency list, lalu melakukan traversal DFS dan BFS mulai dari node A. Hasil traversal menunjukkan urutan kunjungan node sesuai karakteristik masing-masing algoritma, di mana DFS menelusuri cabang sedalam mungkin terlebih dahulu, sedangkan BFS menelusuri node berdasarkan tingkat (level) kedekatan dari node awal.

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

1. Unguided 1

Graph.h

```h
#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 printDFS(Graph &G, adrNode N);
void printBFS(Graph &G, adrNode N);

#endif
```

Graph.cpp

```
#include "graph.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!" << endl;
        return;
    }

    // Buat edge dari N1 ke N2
    adrEdge E1 = allocateEdge(N2);
    E1->next = N1->firstEdge;
    N1->firstEdge = E1;

    // Karena undirected graph, buat edge balik dari N2 ke N1
    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 << " ";
```

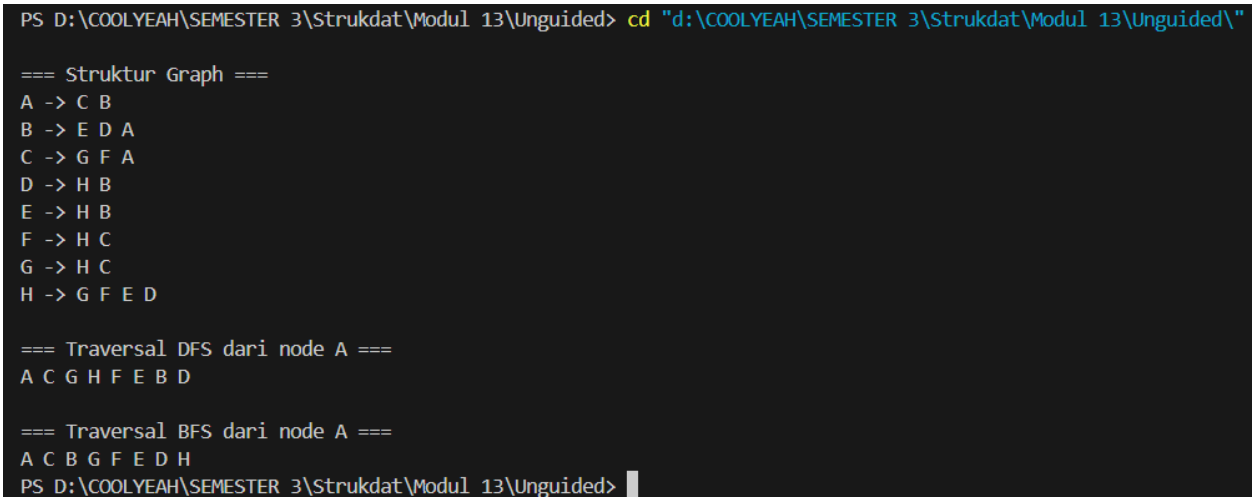
```
            adrEdge E = curr->firstEdge;
            while (E != NULL) {
                if (E->node->visited == 0) {
                    Q.push(E->node);
                }
                E = E->next;
            }
        }
    }
}
```

Main.cpp

```
#include "graph.h"
#include "graph.cpp"

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

    // Tambah node sesuai gambar ilustrasi (A, B, C, D, E, F, G, H)
    insertNode(G, 'H');
    insertNode(G, 'G');
    insertNode(G, 'F');
    insertNode(G, 'E');
    insertNode(G, 'D');
    insertNode(G, 'C');
    insertNode(G, 'B');
    insertNode(G, 'A');

    // Hubungkan node sesuai dengan gambar ilustrasi di modul
    // Dari gambar: A terhubung dengan B dan C
    connectNode(G, 'A', 'B');
    connectNode(G, 'A', 'C');

    // B terhubung dengan D dan E
    connectNode(G, 'B', 'D');
    connectNode(G, 'B', 'E');

    // C terhubung dengan F dan G
    connectNode(G, 'C', 'F');
    connectNode(G, 'C', 'G');

    // D, E, F, G semuanya terhubung ke H
    connectNode(G, 'D', 'H');
    connectNode(G, 'E', 'H');
    connectNode(G, 'F', 'H');
    connectNode(G, 'G', 'H');
```

```
    cout << "\n=== Struktur Graph ===" << endl;
    printInfoGraph(G);

    cout << "\n=== Traversal DFS dari node A ===" << endl;
    resetVisited(G);
    printDFS(G, findNode(G, 'A'));

    cout << "\n\n=== Traversal BFS dari node A ===" << endl;
    resetVisited(G);
    printBFS(G, findNode(G, 'A'));

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

Screenshoot Output



Deskripsi:

Kurang lebih sama seperti yang Guided. Program di atas merupakan implementasi struktur data graph tidak berarah (undirected graph). Graph direpresentasikan dengan sebuah struct Graph yang menyimpan pointer ke node pertama. Setiap node (ElmNode) menyimpan informasi berupa karakter (A–H), penanda kunjungan (visited) untuk keperluan traversal, pointer ke edge pertama, serta pointer ke node berikutnya dalam daftar node. Edge (ElmEdge) digunakan untuk merepresentasikan hubungan antar node, di mana setiap edge menunjuk ke node tujuan dan edge selanjutnya. Program menyediakan fungsi untuk membuat graph, mengalokasikan node dan edge, menambahkan node ke dalam graph, mencari node tertentu, serta menghubungkan dua node dengan edge dua arah karena graph bersifat tidak berarah. Selain itu, terdapat fungsi untuk menampilkan struktur graph dalam bentuk adjacency list, mereset status kunjungan node, serta melakukan traversal Depth First Search (DFS) dan Breadth First Search (BFS). Pada fungsi main, graph diinisialisasi dengan delapan node (A sampai H) dan dihubungkan sesuai ilustrasi, di mana A terhubung ke B dan C, B ke D dan E, C ke F dan G, serta D, E, F, dan G terhubung ke H. Program kemudian menampilkan struktur graph,

diikuti hasil traversal DFS dan BFS dari node A. Hasil traversal menunjukkan karakteristik masing-masing algoritma, di mana DFS menelusuri graph secara mendalam mengikuti satu cabang hingga akhir sebelum berpindah cabang lain, sedangkan BFS menelusuri graph secara melebar berdasarkan tingkat kedekatan node dari node awal.

2. Unguided 2

Graph.cpp

```
// untuk unguided nomor 2
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;
    }
}
```

Main.cpp

```
cout << "\n=== DFS Traversal dari node A ===" << endl;
resetVisited(G);
printDFS(G, findNode(G, 'A'));
```

Screenshot Output

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 13\Unguided> cd "d:\COOLYEAH\SEMESTER 3\Strukdat\Modul 13\Unguided\"

=== Struktur Graph ===
A -> C B
B -> E D A
C -> G F A
D -> H B
E -> H B
F -> H C
G -> H C
H -> G F E D

=== DFS Traversal dari node A ===
A C G H F E B D
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 13\Unguided>
```

Deskripsi:

Prosedur printDFS merupakan implementasi algoritma Depth First Search yang digunakan untuk menelusuri graph dengan cara masuk sedalam mungkin terlebih dahulu sebelum mundur ke cabang lainnya. Prosedur ini bekerja secara rekursif, dimana ketika mengunjungi sebuah node, prosedur akan menandai node tersebut sebagai sudah dikunjungi (visited = 1), mencetak informasi node, kemudian secara rekursif mengunjungi semua node tetangga yang belum dikunjungi melalui edge yang terhubung. Algoritma ini menggunakan struktur data stack secara implisit melalui rekursi, sehingga penelusuran akan terus berlanjut ke node-node yang lebih dalam hingga tidak ada lagi node yang dapat dikunjungi, baru kemudian mundur (backtrack) untuk menelusuri cabang lain yang belum dikunjungi. Sebagai contoh, jika terdapat graph dengan struktur A → B → D dan A → C, maka DFS akan mengunjungi A, kemudian B, lalu D sampai mentok, baru kembali ke A untuk mengunjungi C.

3. Unguided 3

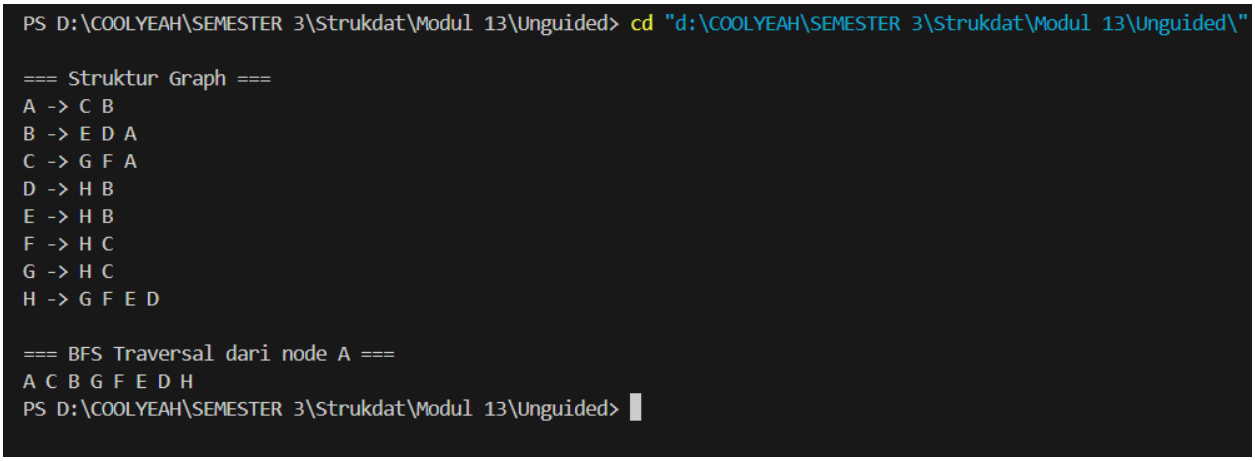
Graph.cpp

```cpp
/* UNTUK UNGUIDED NOMOR 3*/
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

```
cout << "\n=== BFS Traversal dari node A ===" << endl;
resetVisited(G);
printBFS(G, findNode(G, 'A'));
```

Screenshot Output



Deskripsi:

Prosedur printBFS merupakan implementasi algoritma Breadth First Search yang menelusuri graph dengan cara mengunjungi semua node pada level atau kedalaman yang sama terlebih dahulu sebelum berpindah ke level berikutnya. Prosedur ini menggunakan struktur data queue untuk menyimpan node-node yang akan dikunjungi, dimulai dengan memasukkan node awal ke dalam queue, kemudian secara iteratif mengambil node dari depan queue, menandainya sebagai visited, mencetak informasinya, dan memasukkan semua node tetangga yang belum dikunjungi ke dalam queue. Dengan menggunakan prinsip FIFO (First In First Out) pada queue, algoritma ini memastikan bahwa semua node pada level yang sama akan diproses terlebih dahulu sebelum melanjutkan ke level berikutnya. Sebagai contoh, pada graph yang sama A → B → D dan A → C, BFS akan mengunjungi A terlebih dahulu, kemudian mengunjungi semua tetangganya yaitu B dan C pada level yang sama, baru kemudian mengunjungi D yang berada pada level lebih dalam, sehingga menghasilkan urutan kunjungan A, B, C, D yang berbeda dengan DFS.

D. Kesimpulan

Berdasarkan hasil pengerjaan praktikum Modul 14 tentang struktur data Graph, baik pada bagian Guided maupun Unguided 1, 2, dan 3, dapat disimpulkan bahwa struktur data graph merupakan bentuk struktur data non-linier yang efektif untuk merepresentasikan hubungan antar data yang kompleks. Implementasi graph menggunakan adjacency list dengan pendekatan multi linked list memungkinkan setiap node memiliki lebih dari satu hubungan (edge) ke node lain, sehingga sesuai untuk memodelkan graph tidak berarah (undirected graph).

E. Rerferensi

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2006-2007/Makalah/Makalah0607-79.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2008-2009/Makalah2008/Makalah0809-097.pdf

https://www.slideshare.net/slideshow/graphstruktur-datapdf/255319673

https://media.neliti.com/media/publications/75571-ID-rekayasa-pembalikan-kode-berorientasi-ob.pdf

https://fikti.umsu.ac.id/algoritma-bfs-breadth-first-search-pengertian-fungsi-dan-cara-kerja/

https://id.scribd.com/document/712484188/tik

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Makalah/Makalah-IF2211-Stima-2024%20(8).pdf

https://abdulkadir.blog.uma.ac.id/wp-content/uploads/sites/365/2019/02/Modul-Praktikum-Algoritma-Struktur-Data-Merge.pdf

https://jurnal.polgan.ac.id/index.php/jmp/article/download/12340/1537

https://jurnal.stkippersada.ac.id/jurnal/index.php/jutech/article/download/4263/pdf