LAPORAN PRAKTIKUM
Modul 14
Graph

Disusun Oleh:
Zhafir Zaidan Avail (2311104059)
S1-SE-07-2

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

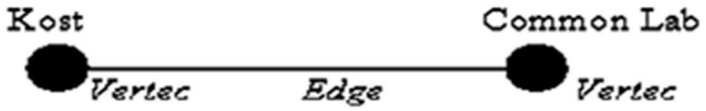
1. Tujuan

1. Memahami konsep graph
2. Mengimplementasikan graph dengan menggunakan pointer.

2. Landasan Teori

1. Pengertian

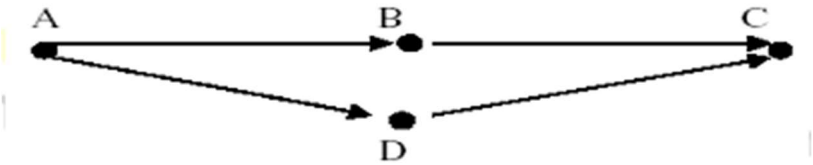
Graph merupakan himpunan tidak kosong dari node (vertec) dan garis penghubung (edge). Contoh sederhana tentang graph, yaitu antara Tempat Kost Anda dengan Common Lab. Tempat Kost Anda dan Common Lab merupakan node (vertec). Jalan yang menghubungkan tempat Kost dan Common Lab merupakan garis penghubung antara keduanya (edge).



2. Jenis Jenis Graph

i. Graph Berarah

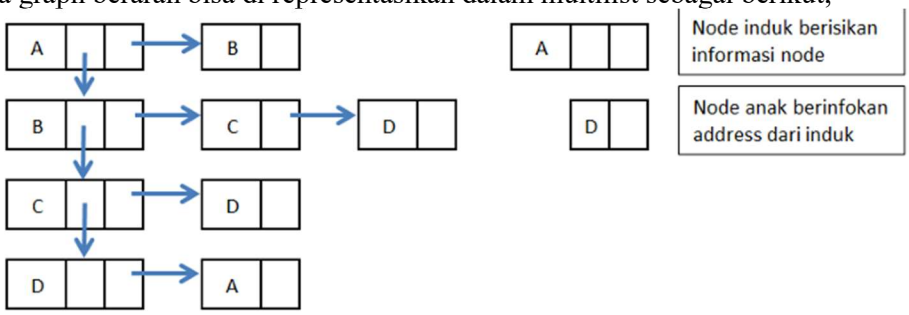
Merupakan graph dimana tiap node memiliki edge yang memiliki arah, kemana node tersebut dihubungkan.



1. Representasi Graph

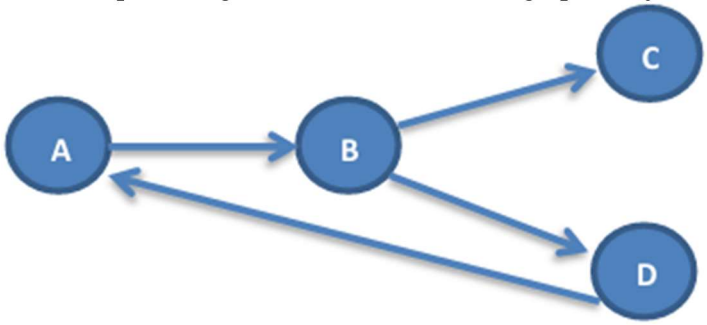
Pada dasarnya representasi dari graph berarah sama dengan graph tak-berarah. Perbedaannya apabila graph tak-berarah terdapat node A dan node B yang terhubung, secara otomatis terbentuk panah bolak balik dari A ke B dan B ke A. Pada Graph berarah node A terhubung dengan node B, belum tentu node B terhubung dengan node A.

Pada graph berarah bisa di representasikan dalam multilist sebagai berikut,



Dalam praktikum ini untuk merepresentasikan graph akan menggunakan multilist. Karena sifat list yang dinamis.

Dari multilist di atas apabila digambarkan dalam bentuk graph menjadi :



2. Topological Sort

Diberikan urutan partial dari elemen suatu himpunan, dikehendaki agar elemen yang terurut parsial tersebut mempunyai keterurutan linier. Contoh dari keterurutan parsial banyak dijumpai dalam kehidupan sehari-hari, misalnya:

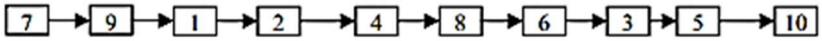
1. Dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisite mata pelajaran lain. Urutan linier adalah urutan untuk seluruh mata pelajaran dalamkurikulum.
2. Dalam suatu proyek, suatu pekerjaan harus dikerjakan lebih dulu dari pekerjaan lain (misalnya membuat fondasi harus sebelum dinding, membuat dinding harus sebelum pintu. Namun pintu dapat dikerjakan bersamaan dengan jendela, dsb.
3. Dalam sebuah program Pascal, pemanggilan prosedur harus sedemikian rupa, sehingga peletakan prosedur pada teks program harus seuai dengan urutan (partial) pemanggilan.

Dalam pembuatan tabel pada basis data, tabel yang di-refer oleh tabel lain harus dideklarasikan terlebih dulu. Jika suatu aplikasi terdiri dari banyak tabel, maka urutan pembuatan tabel harus sesuai dengan definisinya. Jika X < Y adalah simbol untuk X "sebelum" Y, dan keterurutan partial digambarkan sebagai graf, maka graf sebagai berikut :

akan dikatakan mempunyai keterurutan partial

1<2 2<4 4<6 2<10 4<8 6<3 1<3
3<5 5<8 7<5 7<9 9<4 9<10

Dan yang SALAH SATU urutan linier adalah graf sebagai berikut :



Kenapa disebut salah satu urutan linier ? Karena suatu urutan partial akan mempunyai banyak urutan linier yang mungkin dibentuk dari urutan partial tersebut. Elemen yang membentuk urutan linier disebut sebagai "list". Proses yang dilakukan untuk mendapatkan urutan linier :

1. Andaikata item yang mempunyai keterurutan partial adalah anggota himpunan S.
2. Pilih salah satu item yang tidak mempunyai predecessor, misalnya X. Minimal adasatu elemen semacam ini. Jika tidak, maka akan looping.
3. Hapus X dari himpunan S, dan insert ke dalam list
4. Sisa himpunan S masih merupakan himpunan terurut partial, maka proses 2-3 dapat dilakukan lagi terhadap sisa dari S
5. Lakukan sampai S menjadi kosong, dan list Hasil mempunyai elemen dengan keterurutan linier

Solusi I :

Untuk melakukan hal ini, perlu ditentukan suatu representasi internal. Operasi yang penting adalah memilih elemen tanpa predecessor (yaitu jumlah predecessor elemen sama dengan nol). Maka setiap elemen mempunyai 3 karakteristik : identifikasi, list suksesornya, dan banyaknya predecessor. Karena jumlah elemen bervariasi, maka representasi yang paling cocok adalah list berkait dengan representasi dinamis (pointer). List dari successor direpresentasi pula secara berkait.

Representasi yang dipilih untuk persoalan ini adalah multilist sebagai berikut :

1. List yang digambarkan horisontal adalah list dari banyaknya predecessor setiap item, disebut list "Leader", yang direpresentasi sebagai list yang dicatat alamat elemen pertama dan terakhir (Head-Tail) serta elemen terurut menurut key. List ini dibentuk dari

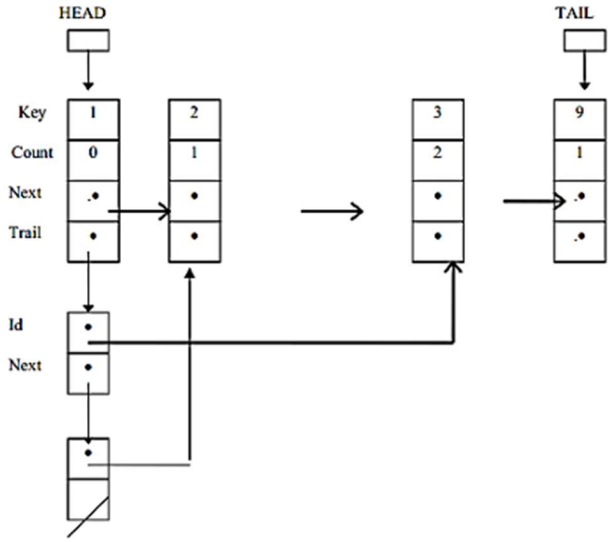
pembacaan data. Untuk setiap data keterurutan partial X < Y : Jika X dan/atau Y belum ada pada list leader, insert pada Tail dengan metoda search dengan sentinel.

2. List yang digambarkan vertikal (ke bawah) adalah list yang merupakan indirect addressing ke setiap predecessor, disebut sebagai "Trailer". Untuk setiap elemen list Leader X, list dari suksesornya disimpan sebagai elemen list Trailer yang setiapelemennya berisi alamat dari successor. Penyisipan data suatu successor (X < Y), dengan diketahui X, maka akan dilakukan dengan InsertFirst alamat Y sebagai elemen list Trailer dengan key X.

Algoritma secara keseluruhan terdiri dari dua pass :

1. Bentuk list leader dan Trailer dari data keterurutan partial : baca pasangan nilai (X < Y). Temukan alamat X dan Y (jika belum ada sisipkan), kemudian dengan mengetahui alamat X dan Y pada list Leader, InsertFirst alamat Y sebagai trailer X.
2. Lakukan topological sort dengan melakukan search list Leader dengan jumlah predecessor=0, kemudian insert sebagai elemen list linier hasil pengurutan.

Ilustrasi umum dari list Leader dan Trailer untuk representasi internal persoalan topological sorting adalah sebagai berikut.



Solusi II : pendekatan "fungsional" dengan list linier sederhana.

Pada solusi ini, proses untuk mendapatkan urutan linier diterjemahkan secara fungsional, dengan representasi sederhana. Graf partial dinyatakan sebagai list linier dengan representasi fisik First-Last dengan dummy seperti representasi pada Solusi I. dengan elemen yang terdiri dari . Contoh: sebuah elemen bernilai artinya 1 adalah predecessor dari 2.

Representasi Topological Sort

Representasi graph untuk topological sort sama dengan graph berarah pada umumnya.

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

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

struct ElmNode{
 infoGraph info;
 int Visited;
 int Pred;
 adrEdge firstEdge;
 adrNode Next;
};
struct ElmEdge{
```

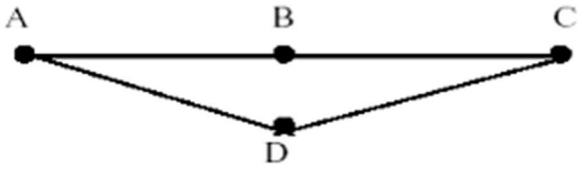
```
 adrNode Node;
 adrEdge Next;
};
struct Graph {
 adrNode First;
};

adrNode AllocateNode (infoGraph X);
adrEdge AllocateEdge (adrNode N);
void CreateGraph (Graph &G);
void InsertNode (Graph &G, infoGraph X);
void DeleteNode (Graph &G, infoGraph X);
void ConnectNode (adrNode N1, adrNode N2);
void DisconnectNode (adrNode N1, adrNode N2);
adrNode FindNode (Graph G, infoGraph X);
adrEdge FindEdge (adrNode N, adrNode NFind);
void PrintInfoGraph (Graph G);
void PrintTopologicalSort (Graph G);

#endif
```

ii. Graph Tak Berarah

Merupakan graph dimana tiap node memiliki edge yang dihubungkan ke node lain tanpa arah.



3.

Selain arah, beban atau nilai sering ditambahkan pada edge . Misalnya nilai yang merepresentasikan panjang, atau biaya transportasi, dan lain-lain. Hal mendasar lain yang perlu diketahui adalah, suatu node A dikatakan bertetangga dengan node B jika antara node A dan node B dihubungkan langsung dengan sebuah edge.

Misalnya:

Dari gambar contoh graph pada halaman sebelumnya dapat disimpulkan bahwa: A bertetangga dengan B, B bertetangga dengan C, A tidak bertetangga dengan C, B tidak bertetangga dengan D.

Masalah ketetanggaan suatu node dengan node yang lain harus benar-benar diperhatikan dalam implementasi pada program. Ketetanggaan dapat diartikan sebagai keterhubungan antar node yang nantinya informasi ini dibutuhkan untuk melakukan beberapa proses seperti : mencari lintasan terpendek dari suatu node ke node yang lain, pengubahan graph menjadi tree (untuk perancangan jaringan) dan lain-lain.

Tentu anda sudah tidak asing dengan algoritma Djikstra, Kruskal, Prim dsb. Karena waktu praktikum terbatas, kita tidak membahas algoritma tersebut. Di sini anda hanya akan mencoba untuk mengimplementasikan graph dalam program.

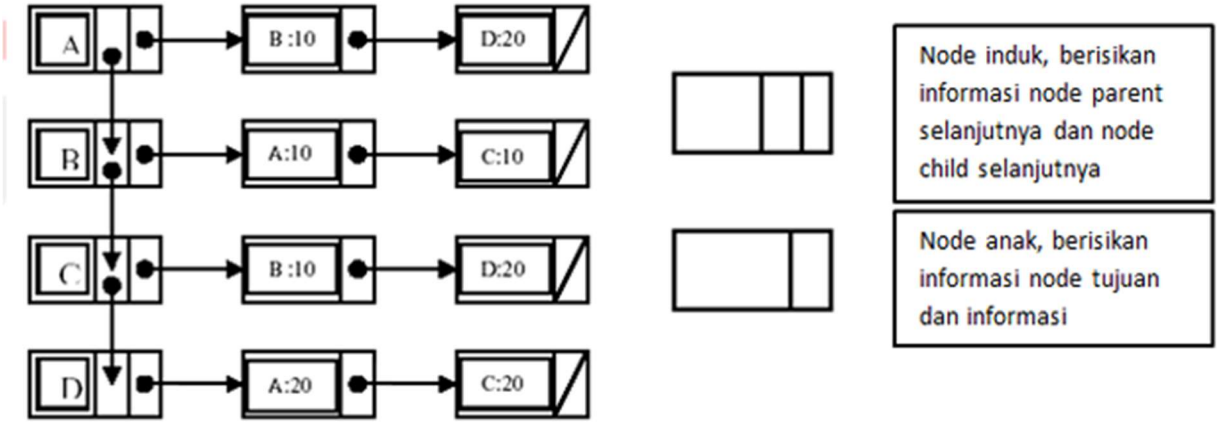
1. Representasi Graph

Dari definisi graph dapat kita simpulkan bahwa graph dapat direpresentasikan dengan Matrik Ketetanggaan (Adjacency Matrices), yaitu matrik yang menyatakan keterhubungan antar node dalam graph. Implementasi matrik ketetanggaan dalam bahasa pemrograman dapat berupa : Array 2 Dimensi dan Multi Linked List. Graph dapat direpresentasikan dengan matrik n x n, dimana n merupakan jumlah node dalam graph tersebut.

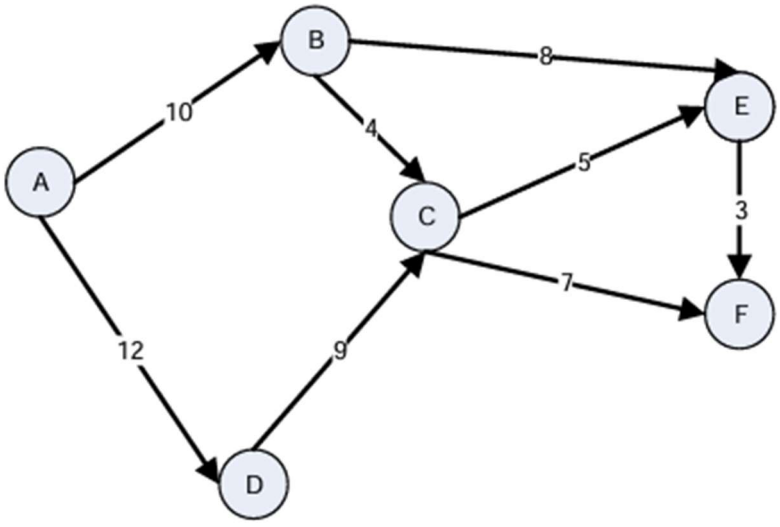
	A	B	C	D
A	-	1	0	1
B	1	-	1	0
C	0	1	-	1
D	1	0	1	-

Keterangan : 1 bertetangga
0 tidak bertetangga

A. Array 2 Dimensi



Dalam praktikum ini untuk merepresentasikan graph akan menggunakan multi list. Karena sifat list yang dinamis, sehinga data yang bisa ditangani bersifat dinamis. Contoh ada sebuah graph yang menggambarkan jarak antar kota:



Gambar multilist-nya sama dengan gambar di atas.
Representasi struktur data graph pada multilist:

```
#ifndef GRAPH_H_INCLUDE
#define GRAPH_H_INCLUDE
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

typedef int infoGraph;
typedef struct ElmNode *adrNode;
typedef struct ElmEdge *adrEdge;
```

```
struct ElmNode{
 infoGraph info;
 int Visited;
 adrEdge firstEdge;
 adrNode Next;
};
struct ElmEdge{
 adrNode Node;
 adrEdge Next;
};
struct Graph {
 adrNode First;
};
```

Berikut adalah contoh fungsi tambah node (addNode) dan prosedur tambah edge (addEdge):

```
// Adds Node
ElmNode addNode (infoGraph a, int b, adrEdge c, adrNode d){
 ElmNode newNode;
 newNode.Info = a ;
 newNode.Visited = b ;
 newNode.firstEdge = c ;
 newNode.Next = d ;
 return newNode;
}

// Adds an edge to a graph
void addEdge(ElmNode newNode){
 ElmEdge newEdge ;
 newEdge.Node = newNode.Next;
 newEdge.Next = newNode.firstEdge;
}
```

Karena representasinya menggunakan multilist maka primitif-primitif graph sama dengan primitif primitif pada multilist. Jadi untuk membuat ADT graph bisa memanfaatkan ADT yang sudah dibuat pada multilist.

B. Multi Linked List

2. Metode Metode Penelusuran Graph

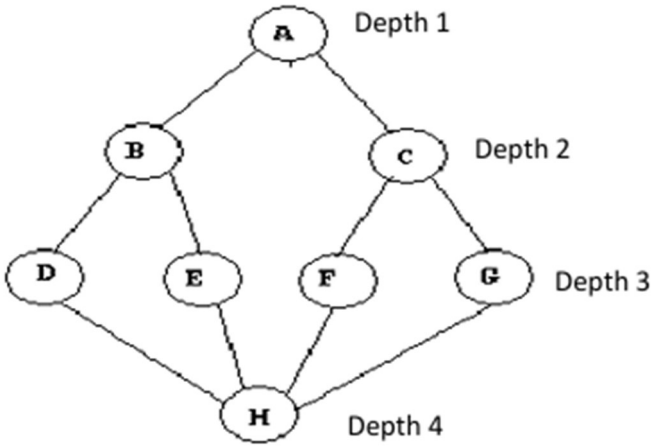
A. Breadth First Search (BFS)\

Cara kerja algoritma ini adalah dengan mengunjungi root (depth 0) kemudian ke depth 1, 2, dan seterusnya. Kunjungan pada masing-masing level dimulai dari kiri ke kanan.

Secara umum, Algoritma BFS pada graph adalah sebagai berikut:

```
Prosedur BFS ( g : graph, start : node )
Kamus
 Q : Queue
 x, w : node
Algoritma
enqueue ( Q, start )
while ( not isEmpty( Q ) ) do
x □ dequeue ( Q )
if ( isVisited( x ) = false ) then
isVisited( x ) □ true
output ( x )
for each node w Є Vx
if ( isVisited( w ) = false ) then
enqueue( Q, w )
```

Perhatikan graph berikut :



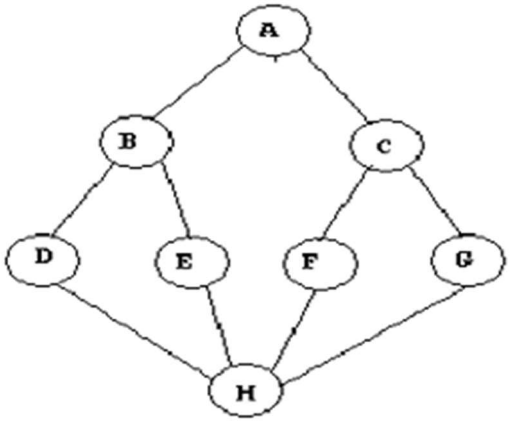
Urutannya hasil penelusuran BFS : A B C D E F G H

B. Depth First Search (DFS)

Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian rekursif ke subtree node tersebut. Secara umum, Algoritma DFS pada graph adalah sebagai berikut:

```
Prosedur BFS ( g : graph, start : node )
Kamus
S : Stack
x, w : node
Algoritma
push ( S, start )
while ( not isEmpty( S ) ) do
x  pop ( S )
if ( isVisited( x ) = false ) then
isVisited( x )  true
output ( x )
for each node w Є Vx
if ( isVisited( w ) = false ) then
push ( S, w )
```

Perhatikan graph berikut :



Urutannya : A B D H E F C G

3. Guided Code:

```
#include <iostream>
#include <queue>

using namespace std;

struct ElmNode;
```

```
struct ElmEdge {
    ElmNode *Node;
    ElmEdge *Next;
};

struct ElmNode {
    char info;
    bool visited;
    ElmEdge *firstEdge;
    ElmNode *Next;
};

struct Graph {
    ElmNode *first;
};

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

void InsertNode(Graph &G, char X) {
    ElmNode *newNode = new ElmNode;
    newNode->info = X;
    newNode->visited = false;
    newNode->firstEdge = NULL;
    newNode->Next = NULL;

    if (G.first == NULL) {
        G.first = newNode;
    } else {
        ElmNode *temp = G.first;
        while (temp->Next != NULL) {
            temp = temp->Next;
        }
        temp->Next = newNode;
    }
}

void ConnectNode(ElmNode *N1, ElmNode *N2) {
    ElmEdge *newEdge = new ElmEdge;
    newEdge->Node = N2;
    newEdge->Next = N1->firstEdge;
    N1->firstEdge = newEdge;
}

void PrintInfoGraph(Graph G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        cout << temp->info << " ";
        temp = temp->Next;
    }
    cout << endl;
}

void ResetVisited(Graph &G) {
    ElmNode *temp = G.first;
    while (temp != NULL) {
        temp->visited = false;
        temp = temp->Next;
    }
}

void PrintDFS(Graph G, ElmNode *N) {
    if (N == NULL) {
        return;
    }
```

```
    N->visited = true;
    cout << N->info << " ";
    ElmEdge *edge = N->firstEdge;
    while (edge != NULL) {
        if (!edge->Node->visited) {
            PrintDFS(G, edge->Node);
        }
        edge = edge->Next;
    }
}

void PrintBFS(Graph G, ElmNode *N) {
    queue<ElmNode*> q;
    q.push(N);
    N->visited = true;

    while (!q.empty()) {
        ElmNode *current = q.front();
        q.pop();
        cout << current->info << " ";

        ElmEdge *edge = current->firstEdge;
        while (edge != NULL) {
            if (!edge->Node->visited) {
                edge->Node->visited = true;
                q.push(edge->Node);
            }
            edge = edge->Next;
        }
    }
}

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');

    ElmNode *A = G.first;
    ElmNode *B = A->Next;
    ElmNode *C = B->Next;
    ElmNode *D = C->Next;
    ElmNode *E = D->Next;
    ElmNode *F = E->Next;
    ElmNode *G1 = F->Next;
    ElmNode *H = G1->Next;

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, G1);
    ConnectNode(D, H);

    cout << "DFS traversal: ";
    ResetVisited(G);
    PrintDFS(G, A);
    cout << endl;
```

```
    cout << "BFS traversal: ";
    ResetVisited(G);
    PrintBFS(G, A);
    cout << endl;

    return 0;
}
```

Output:

```
DFS traversal: A C G F B E D H
BFS traversal: A C B G F E D H
```

4. Unguided

Unguided1:

Code:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <limits>

using namespace std;

int main() {
    int jumlahSimpul;

    cout << "Silakan masukan jumlah simpul (minimal 1): ";
    while (true) {
        cin >> jumlahSimpul;
        if (cin.fail() || jumlahSimpul < 1) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Input tidak valid. Silakan masukkan angka positif: ";
        } else {
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            break;
        }
    }

    vector<string> simpul(jumlahSimpul);

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }

    vector<vector<int>> matriks(jumlahSimpul, vector<int>(jumlahSimpul, 0));

    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
            while (true) {
                cin >> matriks[i][j];
                if (cin.fail()) {
                    cin.clear();
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    cout << "Input tidak valid. Silakan masukkan angka: ";
                } else {
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    break;
                }
            }
        }
    }
```

```
    cout << "\nMatriks Bobot:\n";
    cout << setw(8) << " ";
    for (const auto& s : simpul) {
        cout << setw(8) << s;
    }
    cout << endl;

    for (int i = 0; i < jumlahSimpul; i++) {
        cout << setw(8) << simpul[i];
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << setw(8) << matriks[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Output:

```
Silakan masukan jumlah simpul (minimal 1): 4
Simpul 1: A
Simpul 2: D
Simpul 3: S
Simpul 4: X
A --> A = T
Input tidak valid. Silakan masukkan angka: 3
A --> D = 2
A --> S = 4
A --> X = 6
D --> A = 8
D --> D = 1
D --> S = 5
D --> X = 7
S --> A = 9
S --> D = 3
S --> S = 5
S --> X = 7
X --> A = 3
X --> D = 6
X --> S = 9
X --> X = 1

Matriks Bobot:
               A       D       S       X
       A       3       2       4       6
       D       8       1       5       7
       S       9       3       5       7
       X       3       6       9       1
```

Unguided2:

Code:

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

int main() {
    int numVertices, numEdges;

    // Input jumlah simpul dan jumlah sisi
    cout << "Masukkan jumlah simpul: ";
    cin >> numVertices;
    cout << "Masukkan jumlah sisi: ";
    cin >> numEdges;

    // Deklarasi adjacency matrix
```

```
    vector<vector<int>> adjacencyMatrix(numVertices, vector<int>(numVertices,
0));
    vector<string> vertexLabels(numVertices);

    // Input label simpul
    cout << "Masukkan nama simpul:\n";
    for (int i = 0; i < numVertices; ++i) {
        cout << "Simpul " << i + 1 << ": ";
        cin >> vertexLabels[i];
    }

    // Input sisi antar simpul
    cout << "\nMasukkan pasangan simpul yang terhubung:\n";
    for (int i = 0; i < numEdges; ++i) {
        string from, to;
        cout << "Sisi " << i + 1 << ": ";
        cin >> from >> to;

        int fromIdx = -1, toIdx = -1;
        for (int j = 0; j < numVertices; ++j) {
            if (vertexLabels[j] == from) fromIdx = j;
            if (vertexLabels[j] == to) toIdx = j;
        }

        if (fromIdx != -1 && toIdx != -1) {
            adjacencyMatrix[fromIdx][toIdx] = 1;
            adjacencyMatrix[toIdx][fromIdx] = 1; // Untuk graf tak berarah
        } else {
            cout << "Nama simpul tidak valid. Coba lagi.\n";
            --i;
        }
    }

    // Menampilkan adjacency matrix
    cout << "\nAdjacency Matrix:\n";
    cout << " ";
    for (const auto& label : vertexLabels) {
        cout << "\t" << label;
    }
    cout << "\n";

    for (int i = 0; i < numVertices; ++i) {
        cout << vertexLabels[i];
        for (int j = 0; j < numVertices; ++j) {
            cout << "\t" << adjacencyMatrix[i][j];
        }
        cout << "\n";
    }

    return 0;
}
```

Output:

```
Masukkan jumlah simpul: 6
Masukkan jumlah sisi: 4
Masukkan nama simpul:
Simpul 1: A
Simpul 2: B
Simpul 3: C
Simpul 4: D
Simpul 5: E
Simpul 6: F

Masukkan pasangan simpul yang terhubung:
Sisi 1: A
B
```

```
Sisi 2: C
D
Sisi 3: E
F
Sisi 4: F
A

Adjacency Matrix:
        A       B       C       D       E       F
A       0       1       0       0       0       1
B       1       0       0       0       0       0
C       0       0       0       1       0       0
D       0       0       1       0       0       0
E       0       0       0       0       0       1
F       1       0       0       0       1       0
```

5. Kesimpulan

Graph adalah struktur data yang terdiri dari node (vertex) yang dihubungkan oleh garis penghubung (edge). Graph digunakan untuk merepresentasikan hubungan antar objek, seperti koneksi jaringan, jalan antar lokasi, atau hubungan logis dalam data. Terdapat dua jenis utama graph: **graph berarah** dan **graph tak berarah**. Graph berarah memiliki edge yang menunjukkan arah koneksi antara node, sedangkan graph tak berarah memiliki koneksi dua arah tanpa arah tertentu. Representasi graph dalam struktur data dapat menggunakan adjacency matrix, adjacency list, atau multilist, bergantung pada kebutuhan efisiensi penyimpanan dan operasi.

Topological sort adalah metode untuk mengatur elemen dari graph berarah acyclic (DAG) dalam urutan linier berdasarkan keterurutan parsial. Proses ini sangat relevan dalam berbagai situasi nyata, seperti urutan mata pelajaran dalam kurikulum, langkah kerja proyek, atau urutan pembuatan tabel dalam basis data. Dua pendekatan utama untuk melakukan topological sort adalah dengan menggunakan multilist untuk menangani elemen dinamis atau dengan representasi linier yang sederhana. Pendekatan ini memungkinkan elemen tanpa pendahulu diproses terlebih dahulu hingga semua elemen terurut.

Dalam penelusuran graph, **Breadth-First Search (BFS)** dan **Depth-First Search (DFS)** adalah metode utama yang sering digunakan. BFS menjelajahi graph berdasarkan level atau kedalaman tertentu, dimulai dari node root ke node yang berada pada level lebih dalam, cocok untuk menemukan jalur terpendek. Sebaliknya, DFS menjelajahi graph dengan menyusuri satu jalur sedalam mungkin sebelum kembali, lebih efektif untuk menemukan solusi dengan semua kemungkinan eksplorasi. Kedua metode ini berperan penting dalam berbagai aplikasi graph, seperti penelusuran peta, algoritma permainan, dan analisis jaringan.