

LAPORAN PRAKTIKUM
Modul 14
“GRAPH”



Disusun Oleh:

Ahmad Al - Farizi - 2311104054

Kelas :

S1SE-07-02

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

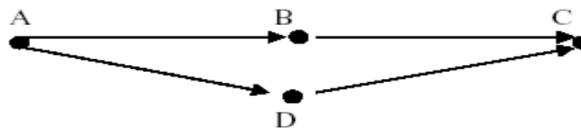
Graph merupakan himpunan tidak kosong dari node (vertex) dan garis penghubung (edge). Contoh sederhana tentang graph, yaitu antara Tempat Kost Anda dengan Common Lab. Tempat Kost Anda dan Common Lab merupakan node (vertex). Jalan yang menghubungkan tempat Kost dan Common Lab merupakan garis penghubung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



Jenis – jenis graph:

1. Graph Berarah (Directed Graph)

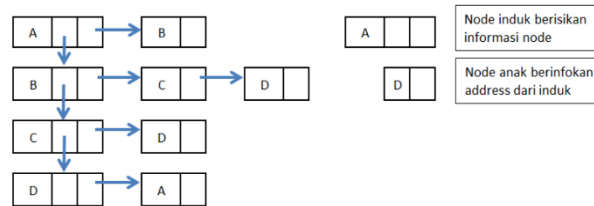
Merupakan graph dimana tiap node memiliki edge yang memiliki arah, kemana node tersebut dihubungkan.



A. Representasi Graph

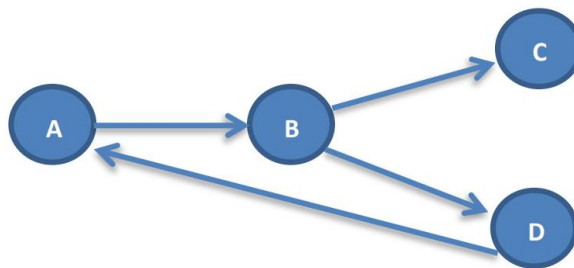
Pada dasarnya representasi dari graph berarah sama dengan graph tak-berarah. Perbedaannya apabila graph tak-berarah terdapat node A dan node B yang terhubung, secara otomatis terbentuk panah bolak balik dari A ke B dan B ke A. Pada Graph berarah node A terhubung dengan node B, belum tentu node B terhubung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



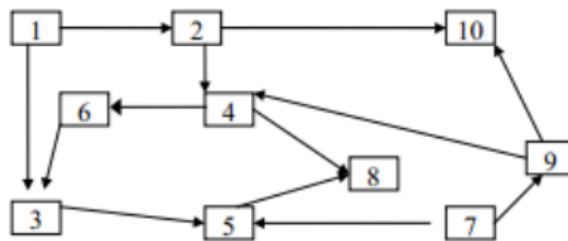
B. Topological Sort

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:

- Dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisite mata pelajaran lain. Urutan linier adalah urutan untuk seluruh mata pelajaran dalam kurikulum
- 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.
- Dalam sebuah program Pascal, pemanggilan prosedur harus sedemikian rupa, sehingga peletakan prosedur pada teks program harus sesuai 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.

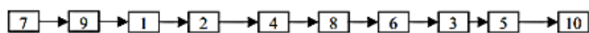
Jika $X < Y$ adalah simbol untuk X “sebelum” Y, dan keterurutan partial digambarkan sebagai graf, maka graf sebagai berikut:



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:

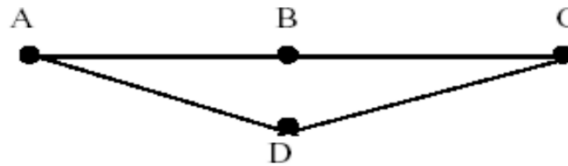


Kenapa disebut salah satu urutan linier? Karena suatu urutan partial akan mempunyai banyak urutan linier yang mungkin dibentuk dari urutan partial tersebut. Elemen yang membentuk urutan linier disebut sebagai “list”. Proses yang dilakukan untuk mendapatkan urutan linier:

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

2. Graph Tidak Berarah (Undirected Graph)

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



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

A. 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

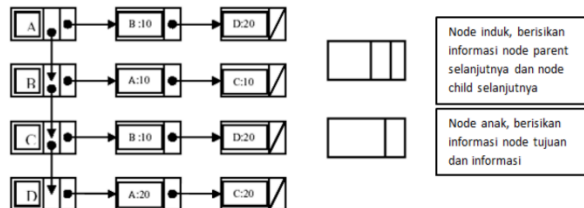
Linked List. Graph dapat direpresentasikan dengan matrik $n \times n$, dimana n merupakan jumlah node dalam graph tersebut.

a. Array 2 Dimensi

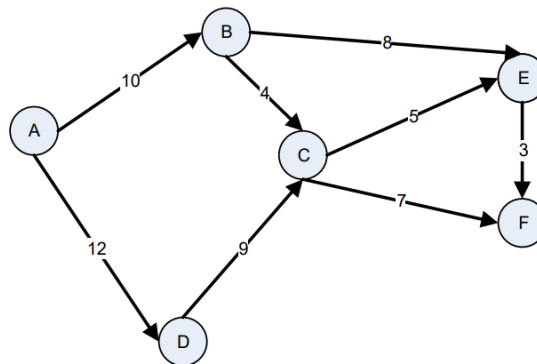
	A	B	C	D
A	-	1	0	1
B	1	-	1	0
C	0	1	-	1
D	1	0	1	-

Keterangan : 1 bertetangga
0 tidak bertetangga

b. Multi Linked List



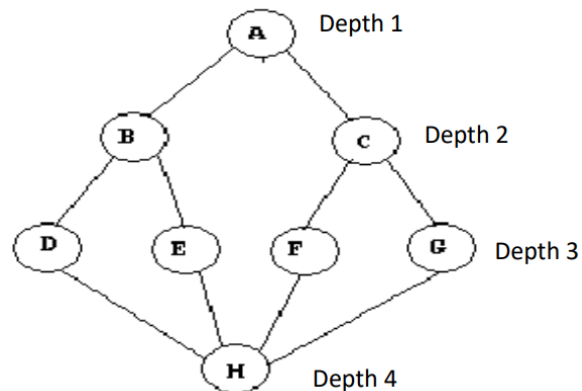
Dalam praktikum ini untuk merepresentasikan graph akan menggunakan multi list. Karena sifat list yang dinamis, sehingga data yang bisa ditangani bersifat dinamis. Contoh ada sebuah graph yang menggambarkan jarak antar kota:



B. Metode – Metode Penelusuran Graph

a. Breadth First Search

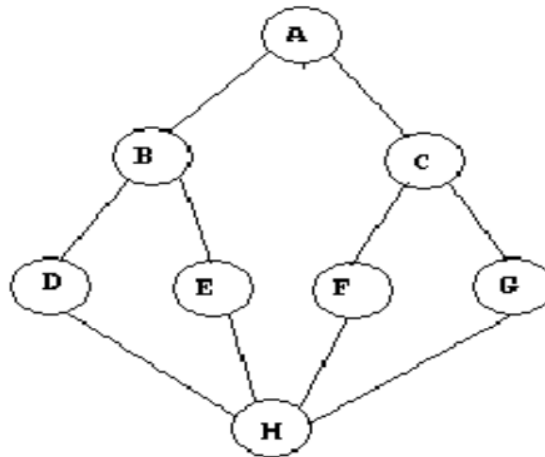
Cara kerja algoritma ini adalah dengan mengunjungi root (depth 0) kemudian ke depth 1, 2, dan seterusnya. Kunjungan pada masing-masing level dimulai dari kiri ke kanan. Perhatikan graph berikut:



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

b. Depth First Search

Cara kerja algoritma ini adalah dengan mengunjungi root, kemudian rekursif ke subtree node tersebut. Perhatikan graph berikut:



Urutannya: A B D H E F C G

3. Guided

1. Program ini mengimplementasikan struktur graf menggunakan daftar berantai di C++. Graf diwakili oleh struktur Graph yang berisi node pertama. Setiap node (ElmNode) menyimpan karakter, status kunjungan, dan pointer ke edge pertama serta node berikutnya. Fungsi CreateGraph menginisialisasi graf, InsertNode menambahkan node, dan ConnectNode menghubungkan dua node. Fungsi PrintDFS dan PrintBFS melakukan traversal graf secara Depth-First Search (DFS) dan Breadth-First Search (BFS) masing-masing, mencetak node yang dikunjungi. Di fungsi main, graf dibentuk dengan beberapa node dan

koneksi, kemudian hasil traversal DFS dan BFS dicetak mulai dari node 'A'.

Kode Program:

```
#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;

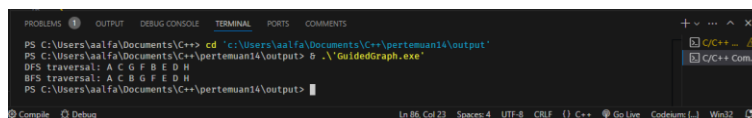
ConnectNode(A, B);
ConnectNode(A, C);
ConnectNode(B, D);
ConnectNode(B, E);
ConnectNode(C, F);
ConnectNode(C, G1);
ConnectNode(D, H);

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

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

return 0;
}
```

Output dari Kode Program:



```
PS C:\Users\aa\fa\Documents\C++> cd 'C:\Users\aa\fa\Documents\C++\pertemuan14\output'
PS C:\Users\aa\fa\Documents\C++\pertemuan14\output> g++ GuidedGraph.exe
BFS traversal: A C B G F E D H
DFS traversal: A C G F B E D H
PS C:\Users\aa\fa\Documents\C++\pertemuan14\output>
```

4. Unguided

1. Nomor 1

Program ini adalah implementasi graf berbasis matriks bobot (adjacency matrix) untuk merepresentasikan jarak antar simpul (kota). Pengguna diminta memasukkan jumlah simpul, nama-nama simpul, dan bobot jarak untuk setiap pasangan simpul, termasuk jarak dari simpul ke dirinya sendiri. Matriks bobot disimpan dalam bentuk array 2 dimensi (`vector<vector<int>>`), di mana setiap elemen mewakili jarak antara dua simpul. Setelah semua data dimasukkan,

program menampilkan matriks bobot dengan nama simpul sebagai header baris dan kolom. Matriks ini mempermudah visualisasi hubungan jarak antar simpul dalam graf.

Kode Program:

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

int main() {
    int jumlahSimpul;

    // Memasukkan jumlah simpul
    cout << "Silakan masukan jumlah simpul: ";
    cin >> jumlahSimpul;

    vector<string> simpul(jumlahSimpul);

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

    // Inisialisasi matriks bobot (adjacency matrix)
    vector<vector<int>> matriks(jumlahSimpul, vector<int>(jumlahSimpul,
        0));

    // Memasukkan bobot antar simpul
    for (int i = 0; i < jumlahSimpul; i++) {
        for (int j = 0; j < jumlahSimpul; j++) {
            cout << simpul[i] << " --> " << simpul[j] << " = ";
```

```

        cin >> matriks[i][j];
    }
}

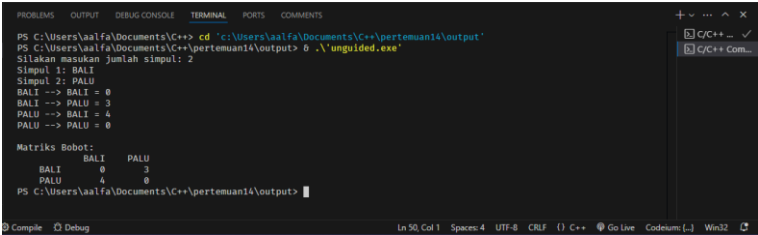
// Menampilkan matriks bobot
cout << "\nMatriks Bobot:\n";
cout << setw(8) << " ";
for (const auto& s : simpul) {
    cout << setw(8) << s;
}
cout << endl;

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 dari Kode Program:



```

PS C:\Users\laalfa\Documents\C++> cd 'c:\Users\laalfa\Documents\C++\pertemuan14\output'
PS C:\Users\laalfa\Documents\C++\pertemuan14\output> 0 .\unguided.exe
Silakan masukan jumlah simpul: 2
Simpul 1: BALI
Simpul 2: PALU
BALI --> BALI = 0
BALI --> PALU = 3
PALU --> BALI = 4
PALU --> PALU = 0

Matriks Bobot:
      BALI  PALU
BALI    0    3
PALU    4    0

PS C:\Users\laalfa\Documents\C++\pertemuan14\output>

```

2. Nomor 2

Kode program C++ ini merepresentasikan graf tidak berarah menggunakan adjacency matrix. Program pertama-tama meminta input jumlah

simpul dan jumlah sisi dari pengguna. Selanjutnya, adjacency matrix diinisialisasi dengan ukuran jumlahSimpul x jumlahSimpul dan semua elemennya diatur ke 0. Pengguna kemudian diminta memasukkan pasangan simpul yang terhubung oleh sisi. Setiap pasangan simpul yang dimasukkan diperbarui dalam matrix dengan mengatur nilai 1 pada posisi yang sesuai, mencerminkan hubungan dua arah. Terakhir, program menampilkan adjacency matrix yang telah diperbarui berdasarkan input pengguna.

Kode Program:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int jumlahSimpul, jumlahSisi;
    cout << "Masukkan jumlah simpul: ";
    cin >> jumlahSimpul;
    cout << "Masukkan jumlah sisi: ";
    cin >> jumlahSisi;

    // Inisialisasi adjacency matrix dengan ukuran jumlahSimpul x
    jumlahSimpul dan nilai awal 0
    vector<vector<int>> adjacencyMatrix(jumlahSimpul,
        vector<int>(jumlahSimpul, 0));

    cout << "Masukkan pasangan simpul yang terhubung oleh sisi (contoh: 1
        2):" << endl;
    for (int i = 0; i < jumlahSisi; ++i) {
        int simpul1, simpul2;
        cin >> simpul1 >> simpul2;

        // Kurangi 1 dari simpul1 dan simpul2 untuk mengkonversi ke indeks
```

```

array (dimulai dari 0)

simpul1--;
simpul2--;

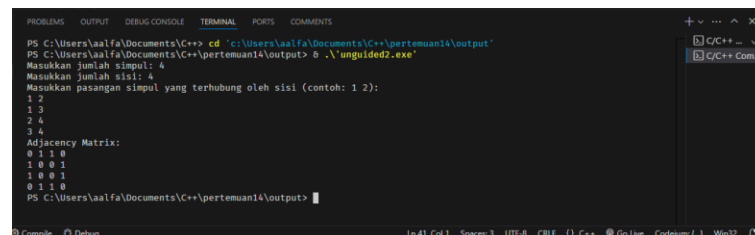
// Karena graf tidak berarah, kita menambahkan sisi di kedua arah
adjacencyMatrix[simpul1][simpul2] = 1;
adjacencyMatrix[simpul2][simpul1] = 1;
}

// Tampilkan adjacency matrix
cout << "Adjacency Matrix:" << endl;
for (int i = 0; i < jumlahSimpul; ++i) {
    for (int j = 0; j < jumlahSimpul; ++j) {
        cout << adjacencyMatrix[i][j] << " ";
    }
    cout << endl;
}

return 0;
}

```

Output Kode Program:



5. Kesimpulan

Graph adalah struktur data yang terdiri dari kumpulan node (vertex) dan garis penghubung (edge), yang dapat berarah (directed graph) atau tidak berarah (undirected graph). Graph berarah memiliki edge dengan arah tertentu, sedangkan graph tidak berarah tidak memiliki arah pada edge-nya. Representasi graph dapat dilakukan dengan matriks ketetanggaan (adjacency matrix) atau multi linked list, tergantung kebutuhan. Graph sering digunakan untuk memodelkan hubungan, seperti jaringan jalan, struktur

proyek, atau basis data, dengan metode traversal seperti Breadth First Search (BFS) dan Depth First Search (DFS) untuk eksplorasi. Topological Sort digunakan untuk menyusun elemen-elemen yang memiliki keterurutan parsial menjadi keterurutan linier. Graph juga berperan penting dalam algoritma pencarian lintasan terpendek dan perancangan jaringan.

