


<b>Nama : Mutiara Novianti Rambe</b> <b>NIM : 064002300029</b>	 <b>Algoritma dan Pemrograman Dasar</b>	<b>Modul 13</b> <b>Nama Dosen:</b> 1. Abdul Rochman 2. Anung B. Ariwibowo
<b>Hari/Tanggal:</b> <b>Kamis/23 Mei 2024</b>		<b>Nama Aslab:</b> 1. Nathanael W. (064002100020) 2. Adrian Alfajri (064002200009)

### MODUL 13 : Graph

**Deskripsi Modul :** Memahami dan menerapkan ilmu struktur data dan algoritma untuk menyelesaikan masalah yang disajikan dengan menggunakan program berbasis bahasa Python.

No.	Elemen Kompetensi	Indikator Kinerja	Halaman
1.	Mampu memahami dan mengimplementasikan Graph pada Python	Membuat dan memahami sebuah program yang menerapkan struktur data Graph.	

#### TEORI SINGKAT

Graph adalah struktur data yang terdiri dari sekumpulan node (disebut juga vertex) dan sekumpulan edge yang menghubungkan node-node tersebut. Graph dapat digunakan untuk merepresentasikan berbagai macam hubungan antar objek, seperti jaringan jalan, jaringan komputer, atau hubungan sosial.

Jenis-jenis Graph:

**Directed Graph (Graph Berarah):** Setiap edge memiliki arah, menghubungkan satu node ke node lainnya dalam satu arah tertentu. Jika ada edge dari node A ke node B, maka  $A \rightarrow B$  tidak sama dengan  $B \rightarrow A$ .

**Undirected Graph (Graph Tak Berarah):** Setiap edge tidak memiliki arah, artinya hubungan antara node bersifat dua arah. Jika ada edge antara node A dan B, maka  $A \leftrightarrow B$  sama dengan  $B \leftrightarrow A$ .

Representasi Graph:

Adjacency List: Setiap node memiliki daftar tetangga yang terhubung dengannya. Representasi ini efisien dalam penggunaan memori dan cepat dalam penambahan dan penghapusan edge.

Adjacency Matrix: Menggunakan matriks dua dimensi untuk menyimpan informasi tentang edge antara node. Matriks ini memudahkan akses cepat untuk memeriksa keberadaan edge, tetapi kurang efisien dalam penggunaan memori terutama untuk graph yang jarang (sparse graph).

### DAFTAR PERTANYAAN

1. Apa yang dimaksud dengan graph dalam struktur data?
2. Sebutkan perbedaan antara directed graph dan undirected graph!
3. Apa kelebihan dan kekurangan dari adjacency matrix dibandingkan dengan adjacency list?

### JAWABAN

1. Graph adalah struktur data yang terdiri dari sekumpulan node (disebut juga vertex) dan sekumpulan edge yang menghubungkan node-node tersebut. Graph dapat digunakan untuk merepresentasikan berbagai macam hubungan antar objek, seperti jaringan jalan, jaringan komputer, atau hubungan sosial.
2. Directed Graph (Graph Berarah): Setiap edge memiliki arah, menghubungkan satu node ke node lainnya dalam satu arah tertentu. Jika ada edge dari node A ke node B, maka  $A \rightarrow B$  tidak sama dengan  $B \rightarrow A$ .  
Undirected Graph (Graph Tak Berarah): Setiap edge tidak memiliki arah, artinya hubungan antara node bersifat dua arah. Jika ada edge antara node A dan B, maka  $A - B$  sama dengan  $B - A$ .

#### 3. Kelebihan Adjacency Matrix:

- Implementasi sederhana dan mudah dipahami
- Cepat untuk mencari tetangga suatu vertex
- Dapat merepresentasikan graf terarah dan tidak terarah

#### Kekurangan Adjacency Matrix:

- Memboroskan ruang, terutama untuk graf yang jarang
- Kurang efisien untuk menambahkan/menghapus edge
- Tidak efisien untuk graf yang jarang

#### Kelebihan Adjacency List:

- Efisien untuk ruang, terutama untuk graf yang jarang
- Efisien untuk menambahkan/menghapus edge
- Efisien untuk graf yang jarang

### Kekurangan Adjacency List:

- Implementasi lebih rumit
- Lambat untuk mencari tetangga suatu vertex (untuk graf besar)
- Hanya dapat merepresentasikan graf tidak terarah

### LAB SETUP

Hal yang harus disiapkan dan dilakukan oleh praktikan untuk menjalankan praktikum modul ini, antara lain:

1. Menyiapkan IDE untuk membangun program python (Spyder, Sublime, VSCode, dll);
2. Python sudah terinstal dan dapat berjalan dengan baik di laptop masing-masing;
3. Menyimpan semua dokumentasi hasil praktikum pada laporan yang sudah disediakan.

### ELEMEN KOMPETENSI I

**Deskripsi** : Mampu membuat program tentang graph sesuai perintah yang ada.

**Kompetensi Dasar** : Membuat program yang mengimplementasikan graph menggunakan adjacency list dan matrix dengan operasi penambahan dan penghapusan vertex serta edge.

### LATIHAN 1

Penambahan dan Penghapusan Vertex serta Edge dengan Adjacency List

1. Buat fungsi untuk menambahkan vertex ke dalam graph.

Code Clip:

```
def tambah_vertex(self, vertex):  
    if vertex not in self.graph:  
        self.graph[vertex] = []  
        # Tambahkan untuk penambahan vertex  
    else:  
        # Tambahkan untuk menginformasikan vertex sudah ada
```

2. Buat fungsi untuk menambahkan edge antara dua vertex.

Code Clip:

```
def tambah_edge(self, vertex1, vertex2):
    if vertex1 in self.graph and vertex2 in self.graph:
        if vertex2 not in self.graph[vertex1]:
            self.graph[vertex1].append(vertex2)
            # Tambahkan vertex2 ke adjacency list dari vertex1
        else:
            # Tambahkan bahwa edge sudah ada
    else:
        IndentationError: expected an indented block after
        # Tambahkan bahwa vertex tidak ada
```

3. Buat fungsi untuk menghapus vertex dari graph.

Code Clip:

```
def hapus_vertex(self, vertex):
    if vertex in self.graph:
        for adjacent in self.graph[vertex]:
            self.graph[adjacent].remove(vertex)
            # Hapus referensi vertex dari adjacency list vertex lain
        self.graph.pop(vertex)
        # Konfirmasi penghapusan vertex
    else:
        # Informasikan vertex tidak ada
```

4. Buat fungsi untuk menghapus edge antara dua vertex.

Code Clip:

```
def hapus_edge(self, vertex1, vertex2):
    if vertex1 in self.graph and vertex2 in self.graph[vertex1]:
        self.graph[vertex1].remove(vertex2)
        # Hapus vertex2 dari adjacency list vertex1
        self.graph[vertex2].remove(vertex1)
        # Konfirmasi penghapusan edge
    else:
        # Informasikan edge tidak ada
```

5. Buat fungsi untuk menampilkan adjacency list dari graph.

Code Clip:

```
def tampilkan_graph(self):  
    for vertex, edges in self.graph.items():  
        print(f"{vertex}: {edges}")  
        # Tampilkan adjacency list setiap vertex
```

### Preview Output (Latihan 1)

```
Menu Operasi Graph  
1. Tambah Vertex  
2. Hapus Vertex  
3. Tambah Edge  
4. Hapus Edge  
5. Tampilkan Graph  
6. Keluar  
Pilih operasi: 1  
Masukkan vertex yang ingin ditambahkan: A  
Vertex A ditambahkan.  
  
Menu Operasi Graph  
1. Tambah Vertex  
2. Hapus Vertex  
3. Tambah Edge  
4. Hapus Edge  
5. Tampilkan Graph  
6. Keluar  
Pilih operasi: 1  
Masukkan vertex yang ingin ditambahkan: B  
Vertex B ditambahkan.
```

```
Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 3
Masukkan vertex pertama: A
Masukkan vertex kedua: B
Edge ditambahkan antara A dan B.
```

```
Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 5
A: ['B']
B: ['A']
```

```
Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 4
Masukkan vertex pertama: A
Masukkan vertex kedua: B
Edge dihapus antara A dan B.
```

```
Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 5
B: []
A: []
```

```
Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 2
Masukkan vertex yang ingin dihapus: A
Vertex A dihapus.

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 5
B: []
```

## Source Code

```
class Graph:
    def __init__(self):
        self.graph = {}

    def tambah_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []
            print(f"Vertex {vertex} ditambahkan.")
        else:
            print(f"Vertex {vertex} sudah ada.")

    def tambah_edge(self, vertex1, vertex2):
        if vertex1 in self.graph and vertex2 in self.graph:
            if vertex2 not in self.graph[vertex1]:
                self.graph[vertex1].append(vertex2)
                self.graph[vertex2].append(vertex1) # Jika graf adalah graf tak berarah
                print(f"Edge ditambahkan antara {vertex1} dan {vertex2}.")
            else:
                print(f"Edge dari {vertex1} ke {vertex2} sudah ada.")
        else:
            print(f"Vertex {vertex1} atau {vertex2} tidak ada di dalam graph.")

    def hapus_vertex(self, vertex):
        if vertex in self.graph:
            for adjacent in list(self.graph[vertex]):
                self.graph[adjacent].remove(vertex)
```

```

        self.graph.pop(vertex)
        print(f"Vertex {vertex} dihapus.")
    else:
        print(f"Vertex {vertex} tidak ditemukan.")

def hapus_edge(self, vertex1, vertex2):
    if vertex1 in self.graph and vertex2 in self.graph[vertex1]:
        self.graph[vertex1].remove(vertex2)
        self.graph[vertex2].remove(vertex1) # Jika graf adalah graf tak berarah
        print(f"Edge dari {vertex1} ke {vertex2} dihapus.")
    else:
        print(f"Edge dari {vertex1} ke {vertex2} tidak ditemukan.")

def tampilkan_graph(self):
    for vertex, edges in self.graph.items():
        print(f"{vertex}: {edges}")

def menu():
    graph = Graph()
    while True:
        print("\nMenu Operasi Graph")
        print("1. Tambah Vertex")
        print("2. Hapus Vertex")
        print("3. Tambah Edge")
        print("4. Hapus Edge")
        print("5. Tampilkan Graph")
        print("6. Keluar")
        pilihan = input("Pilih operasi: ")

        if pilihan == '1':
            vertex = input("Masukkan vertex yang ingin ditambahkan: ")
            graph.tambah_vertex(vertex)
        elif pilihan == '2':
            vertex = input("Masukkan vertex yang ingin dihapus: ")
            graph.hapus_vertex(vertex)
        elif pilihan == '3':
            vertex1 = input("Masukkan vertex pertama: ")
            vertex2 = input("Masukkan vertex kedua: ")
            graph.tambah_edge(vertex1, vertex2)
        elif pilihan == '4':
            vertex1 = input("Masukkan vertex pertama: ")
            vertex2 = input("Masukkan vertex kedua: ")
            graph.hapus_edge(vertex1, vertex2)
        elif pilihan == '5':
            graph.tampilkan_graph()
        elif pilihan == '6':

```



```

        break
    else:
        print("Pilihan tidak valid, silakan coba lagi.")

if __name__ == "__main__":
    menu()

```

## Screenshot

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 1
Masukkan vertex yang ingin ditambahkan: A
Vertex A ditambahkan.

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 1
Masukkan vertex yang ingin ditambahkan: B
Vertex B ditambahkan.

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 3
Masukkan vertex pertama: A
Masukkan vertex kedua: B
Edge ditambahkan antara A dan B.

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 5
A: ['B']
B: ['A']

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 4
Masukkan vertex pertama: A
Masukkan vertex kedua: B
Edge dari A ke B dihapus.

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 5
A: []
B: []

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 2
Masukkan vertex yang ingin dihapus: A
Vertex A dihapus.

```

```

Menu Operasi Graph
1. Tambah Vertex
2. Hapus Vertex
3. Tambah Edge
4. Hapus Edge
5. Tampilkan Graph
6. Keluar
Pilih operasi: 5
B: []

```

**LATIHAN 2**

Penambahan dan Penghapusan Vertex serta Edge dengan Adjacency Matrix.

Buat program yang memungkinkan operasi penambahan dan penghapusan vertex serta edge dalam graph menggunakan adjacency matrix.

1. Buat fungsi untuk inialisasi jumlah vertex di adjacency matrix.

Code Clip:

```
class GraphMatrix:
    def __init__(self, num_vertices):
        self.num_vertices = num_vertices
        self.graph = [[0] * num_vertices for _ in range(num_vertices)]
        # Inisialisasi matriks dengan ukuran num_vertices x num_vertices
```

2. Buat fungsi untuk menambahkan edge antar vertex.

Code Clip:

```
def tambah_edge(self, vertex1, vertex2):
    if 0 <= vertex1 < self.num_vertices and 0 <= vertex2 < self.num_vertices:
        # Set nilai matriks [vertex1][vertex2] dan [vertex2][vertex1] menjadi 1
        print(f"Edge ditambahkan antara {vertex1} dan {vertex2}.")
    else:
        print("Vertex tidak valid.")
```

3. Buat fungsi untuk menghapus edge antar vertex.

Code Clip:

```
def hapus_edge(self, vertex1, vertex2):
    if 0 <= vertex1 < self.num_vertices and 0 <= vertex2 < self.num_vertices:
        # Set nilai matriks [vertex1][vertex2] dan [vertex2][vertex1] menjadi 0
        print(f"Edge dihapus antara {vertex1} dan {vertex2}.")
    else:
        print("Vertex tidak valid.")
```

**Preview Output**

```

Masukkan jumlah vertex: 3

Menu Operasi Graph (Adjacency Matrix)
1. Tambah Edge
2. Hapus Edge
3. Tampilkan Graph
4. Keluar
Pilih operasi: 1
Masukkan index vertex pertama: 1
Masukkan index vertex kedua: 2
Edge ditambahkan antara 1 dan 2.

Menu Operasi Graph (Adjacency Matrix)
1. Tambah Edge
2. Hapus Edge
3. Tampilkan Graph
4. Keluar
Pilih operasi: 3
  A B C
A  0 0 0
B  0 0 1
C  0 1 0

```

## Source Code

```

class GraphMatrix:

    def __init__(self, num_vertices):
        self.num_vertices = num_vertices
        self.graph = [[0 for _ in range(num_vertices)] for _ in
range(num_vertices)]

    def tambah_edge(self, vertex1, vertex2):
        if vertex1 < self.num_vertices and vertex2 < self.num_vertices:
            self.graph[vertex1][vertex2] = 1
            self.graph[vertex2][vertex1] = 1
            print("Edge ditambahkan antara", vertex1, "dan", vertex2)
        else:
            print("Vertex tidak valid.")

    def hapus_edge(self, vertex1, vertex2):
        if vertex1 < self.num_vertices and vertex2 < self.num_vertices:
            self.graph[vertex1][vertex2] = 0
            self.graph[vertex2][vertex1] = 0

```

```
        print("Edge dihapus antara", vertex1, "dan", vertex2)
    else:
        print("Vertex tidak valid.")

    def tampilkan_graph(self):
        for i in range(self.num_vertices):
            for j in range(self.num_vertices):
                print(self.graph[i][j], end=" ")
            print()

def main():
    num_vertices = int(input("Masukkan jumlah vertex: "))

    graph = GraphMatrix(num_vertices)

    while True:
        print("\nMenu Operasi Graph (Adjacency Matrix)")
        print("1. Tambah Edge")
        print("2. Hapus Edge")
        print("3. Tampilkan Graph")
        print("4. Keluar")

        operasi = int(input("Pilih operasi: "))

        if operasi == 1:
            vertex1 = int(input("Masukkan index vertex pertama: "))
            vertex2 = int(input("Masukkan index vertex kedua: "))
            graph.tambah_edge(vertex1, vertex2)
        elif operasi == 2:
            vertex1 = int(input("Masukkan index vertex pertama: "))
            vertex2 = int(input("Masukkan index vertex kedua: "))
            graph.hapus_edge(vertex1, vertex2)
        elif operasi == 3:
            graph.tampilkan_graph()
        elif operasi == 4:
            break
        else:
            print("Operasi tidak valid.")

if __name__ == "__main__":
    main()
```

Screenshot

```
Masukkan jumlah vertex: 3

Menu Operasi Graph (Adjacency Matrix)
1. Tambah Edge
2. Hapus Edge
3. Tampilkan Graph
4. Keluar
Pilih operasi: 1
Masukkan index vertex pertama: 1
Masukkan index vertex kedua: 2
Edge ditambahkan antara 1 dan 2

Menu Operasi Graph (Adjacency Matrix)
1. Tambah Edge
2. Hapus Edge
3. Tampilkan Graph
4. Keluar
Pilih operasi: 3
0 0 0
0 0 1
0 1 0

Menu Operasi Graph (Adjacency Matrix)
1. Tambah Edge
2. Hapus Edge
3. Tampilkan Graph
4. Keluar
Pilih operasi: 4
PS C:\Users\mutia>
```

## KESIMPULAN

Melalui praktikum ini saya tahu bahwa Graph adalah struktur data yang terdiri dari sekumpulan node (disebut juga vertex) dan sekumpulan edge yang menghubungkan node-node tersebut. Graph dapat digunakan untuk merepresentasikan berbagai macam hubungan antar objek, seperti jaringan jalan, jaringan komputer, atau hubungan sosial.

Dan melalui praktikum ini saya belajar membuat program menggunakan adjacency list graph dan juga adjacency matrix graph

## CEKLIST

1. Memahami dan mengimplementasikan adjacency list graph pada Python (✓)
2. Memahami dan mengimplementasikan adjacency matrix graph pada Python (✓)

## REFERENSI

<https://www.sciencedirect.com/topics/computer-science/adjacency-list>

<https://www.geeksforgeeks.org/introduction-to-graphs-data-structure-and-algorithm-tutorials/>

<https://www.javatpoint.com/graph-representation>

<https://www.programiz.com/dsa/graph>

<https://towardsdatascience.com/graph-data-structure-cheat-sheet-for-coding-interviews-a38aadf8aa87>