

MUHAMMAD TARMIDZI BARIQ

51422161

3IA11

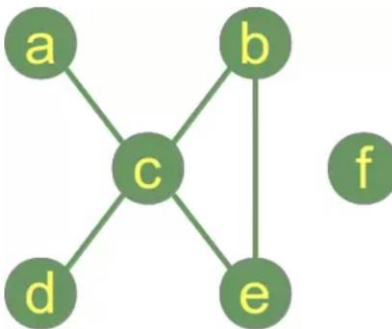
3-Tekrek-M1-Teori Graf (read only)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 Memory: 236.4 MB / 8 GB

Pengenalan Teori Graf

Contoh kasus 1

Dalam contoh kasus pertama kali ini, Kita akan mencoba membangun sebuah graf dengan struktur sebagai berikut.



```
graph LR; a --- c; b --- c; b --- e; c --- d; c --- e; f --- None
```

Dalam merepresentasikan bentuk graf tersebut, kita akan membuat 2 fungsi untuk memudahkan

```
In [1]: ## Blok program ini Anda mencoba mendefinisikan node dan hubungan Antara node dalam bentuk obyek.
graph = { "a" : ["c"],
          "b" : ["c", "e"],
          "c" : ["a", "b", "d", "e"],
          "d" : ["c"],
          "e" : ["c", "b"],
          "f" : []
}
```

Dalam merepresentasikan bentuk graf tersebut, kita akan membuat 2 fungsi untuk memudahkan

```
In [2]: ## Blok program ini Anda mencoba membuat EDGE dari informasi yang sudah Anda definisikan sebelumnya.
def generate_edges(graph):
    edges = []
    for node in graph:
        for neighbour in graph[node]:
            edges.append((node, neighbour))
    return edges

In [3]: print(generate_edges(graph))
[('c', 'a'), ('c', 'b'), ('b', 'e'), ('c', 'e'), ('c', 'd'), ('c', 'b'), ('c', 'a'), ('c', 'd'), ('c', 'e'), ('b', 'e')]

In [4]: ## Blok program ini Anda mendefinisikan sebuah fungsi untuk mengetahui node mana yang tidak memiliki edge
def find_isolated_nodes(graph):
    """ returns a set of isolated nodes. """
    isolated = set()
    for node in graph:
        if not graph[node]:
            isolated.add(node)
    return isolated

In [5]: print(find_isolated_nodes(graph))
{'f'}

Contoh kasus 2

Dalam contoh kasus kedua kali ini, Kita akan mencoba membangun sebuah graf dan mendapatkan informasi lebih detail mengenai graf tersebut. Untuk memudahkan, Kita akan mendefinisikan sebuah Class dalam bahasa pemrograman python, dimana Class ini berisi beberapa fungsi yang bisa kita gunakan untuk mengetahui informasi detail mengenai struktur dari graf yang kita miliki



```
In [6]: """ A Python Class
```



Trending videos 52 hours on Air...



12:59 PM 3/8/2023


```

1

```
graph = { "a" : {"c"},  
          "b" : {"c", "e"},  
          "c" : {"a", "b", "d", "e"},  
          "d" : {"c"},  
          "e" : {"c", "b"},  
          "f" : {}  
        }
```

graph adalah variable

node/simpul dalam graf (yaitu "a", "b", "c", "d", "e", dan "f")

2

```
def generate_edges(graph): ## Mendefinisikan fungsi bernama generate_edges yang menerima  
parameter graph  
  
    edges = [] ## Membuat list kosong bernama edges yang akan digunakan untuk menyimpan  
  
    for node in graph: ## Memulai iterasi/perulangan untuk setiap node (simpul) dalam dictionary graph  
  
        for neighbour in graph[node]: ## Untuk setiap node, melakukan iterasi terhadap semua tetangga  
(neighbour) dari node tersebut  
  
            edges.append({node, neighbour}) ## Menambahkan edge berupa set {node, neighbour} ke  
dalam list edges  
  
    return edges ## Setelah semua iterasi selesai, fungsi mengembalikan list edges yang berisi semua  
edge dalam graf.
```

3

```
print(generate_edges(graph)) ## melakukan print dari fungsi generate_edges dengan parameter graph
```

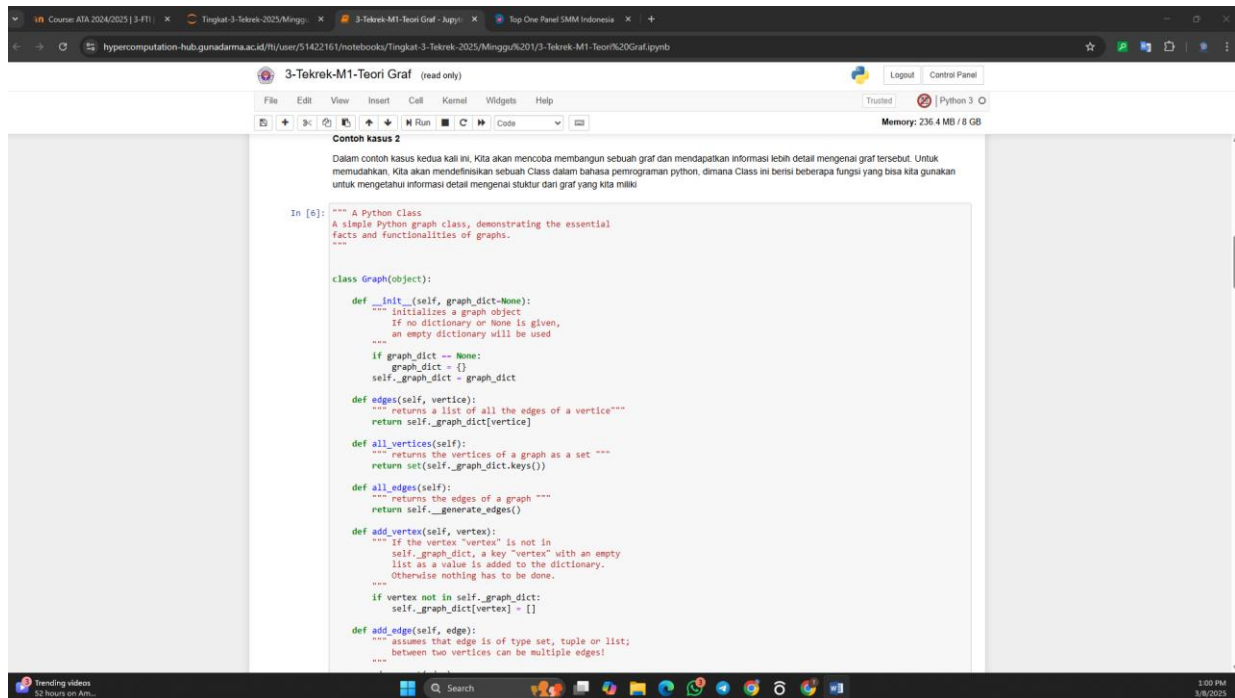
4

```
def find_isolated_nodes(graph): ## Mendefinisikan fungsi bernama find_isolated_nodes yang  
menerima parameter graph  
  
    """ returns a set of isolated nodes. """ ## fungsi ini mengembalikan set berisi node-node yang  
terisolasi.  
  
    isolated = set() ## Membuat set kosong bernama isolated  
  
    for node in graph: ## Memulai iterasi/perulangan untuk setiap node (simpul) dalam dictionary  
graph.  
  
        if not graph[node]: ## Memeriksa apakah nilai (value) yang terkait dengan node saat ini adalah  
kosong. Jika graph[node] adalah set kosong, maka kondisi ini bernilai True.  
  
            isolated.add(node) ## Jika node tidak memiliki tetangga (nilai set kosong), tambahkan node  
tersebut ke dalam set isolated menggunakan metode add().
```

`return isolated` ## Setelah semua iterasi selesai, fungsi mengembalikan set isolated yang berisi semua node terisolasi dalam graf.

5

`print(find_isolated_nodes(graph))` ## melakukan print dari fungsi `find_isolated_nodes` dengan parameter `graph`



The screenshot shows a Jupyter Notebook interface with the title "3-Tekrek-M1-Teori Graf (read only)". The notebook contains a text block and a code cell. The text block describes the purpose of the class and the code cell defines the class `Graph`.

```
Contoh kasus 2

Dalam contoh kasus kedua kali ini, Kita akan mencoba membangun sebuah graf dan mendapatkan informasi lebih detail mengenai graf tersebut. Untuk memudahkan, Kita akan mendefinisikan sebuah Class dalam bahasa pemrograman python, dimana Class ini berisi beberapa fungsi yang bisa kita gunakan untuk mengetahui informasi detail mengenai struktur dan graf yang kita miliki.

In [6]: """ A Python Class
A simple Python graph class, demonstrating the essential
facts and functionalities of graphs.
"""

class Graph(object):

    def __init__(self, graph_dict=None):
        """ Initializes a graph object
        If no dictionary or None is given,
        an empty dictionary will be used
        """
        if graph_dict == None:
            graph_dict = {}
        self._graph_dict = graph_dict

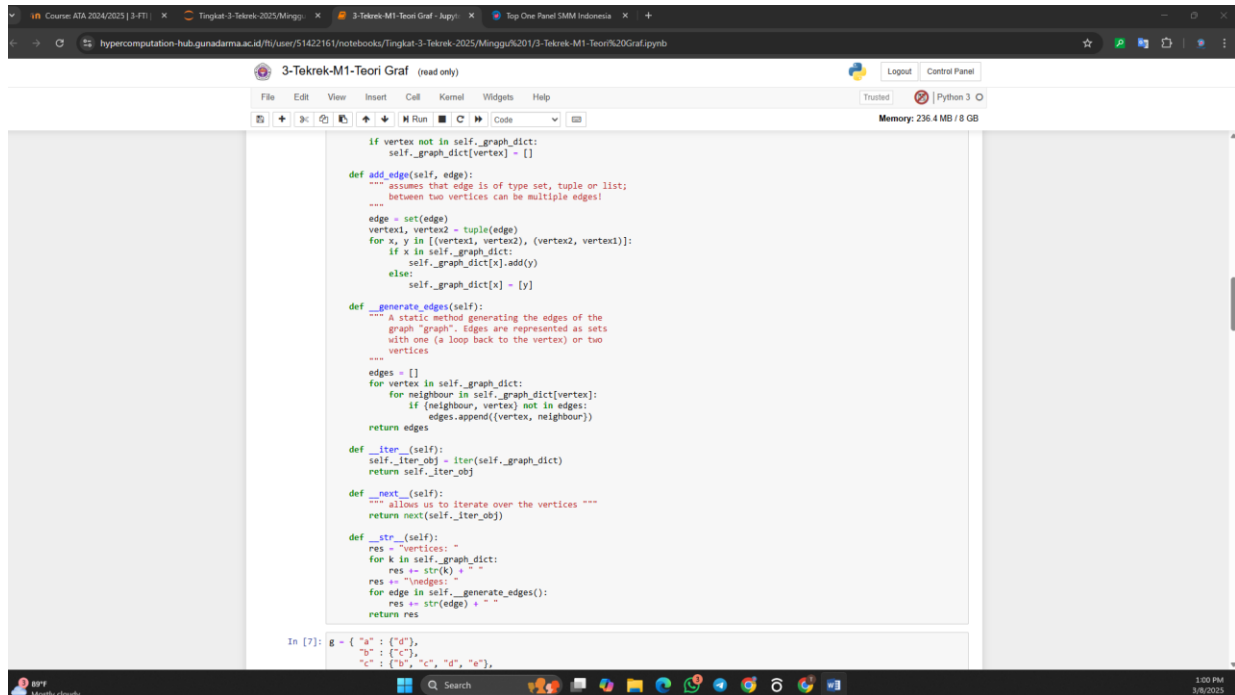
    def edges(self, vertex):
        """ returns a list of all the edges of a vertex"""
        return self._graph_dict[vertex]

    def all_vertices(self):
        """ returns the vertices of a graph as a set """
        return set(self._graph_dict.keys())

    def all_edges(self):
        """ returns the edges of a graph """
        return self._generate_edges()

    def add_vertex(self, vertex):
        """ If the vertex "vertex" is not in
        self._graph_dict, a key "vertex" with an empty
        list as a value is added to the dictionary.
        Otherwise nothing has to be done.
        """
        if vertex not in self._graph_dict:
            self._graph_dict[vertex] = []

    def add_edge(self, edge):
        """ assumes that edge is of type set, tuple or list;
        between two vertices can be multiple edges!
        """
```



The screenshot shows the continuation of the Jupyter Notebook, displaying the implementation of the `Graph` class methods.

```
if vertex not in self._graph_dict:
    self._graph_dict[vertex] = []

def add_edge(self, edge):
    """ assumes that edge is of type set, tuple or list;
    between two vertices can be multiple edges!
    """
    edge = set(edge)
    vertex1, vertex2 = tuple(edge)
    for x, y in [(vertex1, vertex2), (vertex2, vertex1)]:
        if x in self._graph_dict:
            self._graph_dict[x].add(y)
        else:
            self._graph_dict[x] = [y]

def _generate_edges(self):
    """ A static method generating the edges of the
    graph "graph". Edges are represented as sets
    with one (a loop back to the vertex) or two
    vertices
    """
    edges = []
    for vertex in self._graph_dict:
        for neighbour in self._graph_dict[vertex]:
            if (neighbour, vertex) not in edges:
                edges.append((vertex, neighbour))
    return edges

def __iter__(self):
    self._iter_obj = iter(self._graph_dict)
    return self._iter_obj

def next(self):
    """ allows us to iterate over the vertices """
    return next(self._iter_obj)

def __str__(self):
    res = "vertices: "
    for k in self._graph_dict:
        res += str(k) + " "
    res += "\nedges: "
    for edge in self._generate_edges():
        res += str(edge) + " "
    return res

In [7]: g = { "a": [{"d"}],
              "b": [{"c"}],
              "c": [{"b"}, {"c"}, {"d"}, {"a"}],
```

```
hypercomputation-hub.gunadarma.ac.id/hi/user/51422161/notebooks/Tingkat-3-Tekrek-2025/Minggu%201/3-Tekrek-M1-Teori%20Graf.ipynb
3-Tekrek-M1-Teori Graf (read only)
File Edit View Insert Cell Kernel Widgets Help
In [8]: graph = Graph(g)
for vertice in graph:
    print("Edges of vertice {vertice}: ", graph.edges(vertice))
Edges of vertice {vertice}: {'d'}
Edges of vertice {vertice}: {'c'}
Edges of vertice {vertice}: {'c', 'd', 'b', 'e'}
Edges of vertice {vertice}: {'c', 'a'}
Edges of vertice {vertice}: {'c'}
Edges of vertice {vertice}: {}

In [9]: graph.add_edge(("ab", "fg"))
graph.add_edge(("xyz", "bla"))

In [10]: print("")
print("Vertices of graph:")
print(graph.all_vertices())
print("Edges of graph:")
print(graph.all_edges())

Vertices of graph:
{'ab', 'fg', 'd', 'f', 'e', 'a', 'b', 'bla', 'c', 'xyz'}
Edges of graph:
[{'d', 'a'}, {'c', 'b'}, {'c', 'c'}, {'c', 'd'}, {'c', 'e'}, {'fg', 'ab'}, {'xyz', 'bla'}]

Contoh kasus 3
Dalam contoh kasus ketiga kali ini, Kita akan mencoba membangun sebuah graf dengan menggunakan library networkx dan matplotlib

In [11]: import networkx as nx
import matplotlib.pyplot as plt

In [12]: # Creating a Graph
G = nx.Graph() # Right now G is empty

In [13]: G.add_node(1)
G.add_nodes_from([2,3])

In [14]: G.add_edge(1,2)
```

1

```
g = { "a" : {"d"},
```

```
      "b" : {"c"},
```

```
      "c" : {"b", "c", "d", "e"},
```

```
      "d" : {"a", "c"},
```

```
      "e" : {"c"},
```

```
      "f" : {}
```

```
    }
```

graph adalah variable

node/simpul dalam graf (yaitu "a", "b", "c", "d", "e", dan "f")

2

```
graph = Graph(g)
```

```
for vertice in graph:
```

```
    print("Edges of vertice {vertice}: ", graph.edges(vertice))
```

Kode tersebut menggunakan objek Graph yang dibuat dari variabel g (yang tidak ditampilkan dalam kode), kemudian melakukan iterasi untuk setiap vertice (simpul) dalam graf dan mencetak semua edges (tepi) yang terhubung dengan vertice tersebut.

3

```
graph.add_edge({"ab", "fg"})
```

```
graph.add_edge({"xyz", "bla"})
```

menambahkan edge (tepi) baru ke dalam graf.

4

```
print("") ## Mencetak baris kosong sebagai pemisah.
```

```
print("Vertices of graph:") ## Mencetak judul/label untuk daftar simpul.
```

```
print(graph.all_vertices()) ## Memanggil metode all_vertices() dari objek graph dan mencetak hasilnya.  
Metode ini mengembalikan daftar semua simpul yang ada dalam graf.
```

```
print("Edges of graph:") ## Mencetak judul/label untuk daftar tepi.
```

```
print(graph.all_edges()) ## Memanggil metode all_edges() dari objek graph dan mencetak hasilnya.  
Metode ini mengembalikan daftar semua tepi yang ada dalam graf.
```

Contoh kasus 3

Dalam contoh kasus ketiga kali ini, Kita akan mencoba membangun sebuah graf dengan menggunakan library networkx dan matplotlib

```
In [11]: import networkx as nx  
import matplotlib.pyplot as plt
```

```
In [12]: # Creating a Graph  
G = nx.Graph() # Right now G is empty
```

```
In [13]: G.add_node(1)  
G.add_nodes_from([2,3])
```

```
In [14]: G.add_edge(1,2)
```

```
In [15]: e = (2,3)  
G.add_edge(*e) # * unpacks the tuple  
G.add_edges_from([(1,2), (1,3)])
```

```
In [16]: G.nodes()
```

```
Out[16]: NodeView((1, 2, 3))
```

```
In [17]: G.edges()
```

```
Out[17]: EdgeView([(1, 2), (1, 3), (2, 3)])
```

```
In [18]: G.add_edge(1, 2)  
G.add_edge(2, 3)  
G.add_edge(3, 4)  
G.add_edge(1, 4)  
G.add_edge(1, 5)
```

```
In [19]: nx.draw(G, with_labels = True)
```

1

```
import networkx as nx ## Mengimpor library NetworkX yang disingkat sebagai 'nx'
```

```
import matplotlib.pyplot as plt ## Mengimpor submodule pyplot dari library Matplotlib dan  
meningkatkan sebagai 'plt'
```

2

```
G = nx.Graph() ## membuat graph
```

3

```
G.add_node(1) ## Menambahkan satu node dengan label 1 ke dalam graf.
```

```
G.add_nodes_from([2,3]) ## Menambahkan beberapa node sekaligus dari list [2,3], yaitu node 2 dan node 3, ke dalam graf.
```

4

```
G.add_edge(1,2) ## menambahkan satu edge (tepi) ke dalam graf NetworkX yang bernama G.
```

5

```
e = (2,3) ## Membuat tuple yang berisi node 2 dan node 3.
```

```
G.add_edge(*e) ## Menambahkan edge antara node 2 dan node 3. Operator * membuka (unpacks) tuple e sehingga nilainya dijadikan argumen terpisah, sama seperti menulis G.add_edge(2,3).
```

```
G.add_edges_from([(1,2), (1,3)]) ## Menambahkan beberapa edge sekaligus dari list tuple: edge antara node 1 dan 2, serta edge antara node 1 dan 3.
```

6

```
G.nodes() ## metode pada objek graf NetworkX yang mengembalikan objek NodeView (tampilan node)
```

7

```
G.edges() ## metode pada objek graf NetworkX yang mengembalikan objek EdgeView (tampilan edge).
```

8

```
G.add_edge(1, 2)
```

```
G.add_edge(2, 3)
```

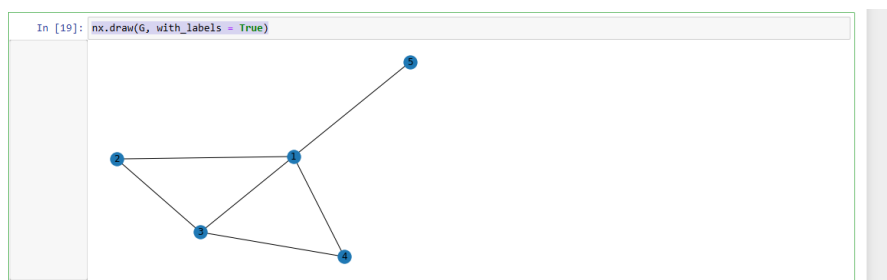
```
G.add_edge(3, 4)
```

```
G.add_edge(1, 4)
```

```
G.add_edge(1, 5)
```

Kode ini menambahkan lima edge (tepi) ke dalam graf NetworkX yang bernama G

9



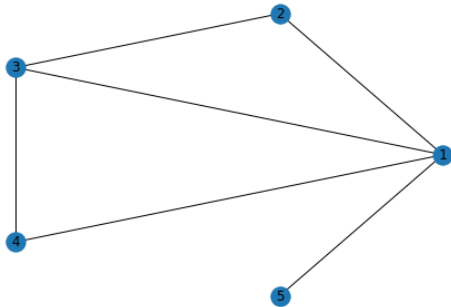
perintah untuk menggambar (visualisasi) graf NetworkX menggunakan Matplotlib.

10

`plt.savefig("contoh-graf-1.png")` ## digunakan untuk menyimpan visualisasi graf yang telah dibuat sebelumnya sebagai file gambar.

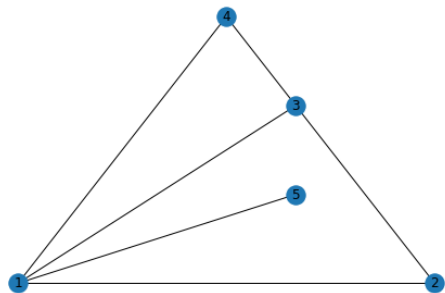
11

```
In [21]: # drawing in circular layout
nx.draw_circular(G, with_labels = True)
```



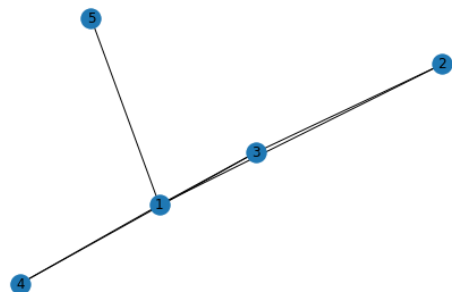
12

```
In [22]: # drawing in planar layout
nx.draw_planar(G, with_labels = True)
```



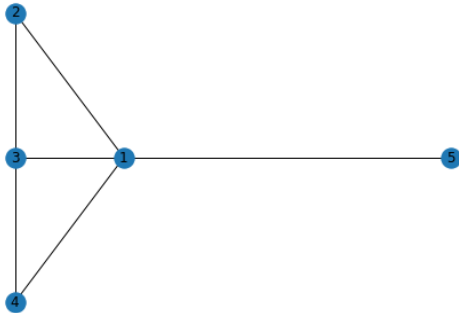
13

```
In [23]: # drawing in random layout
nx.draw_random(G, with_labels = True)
```



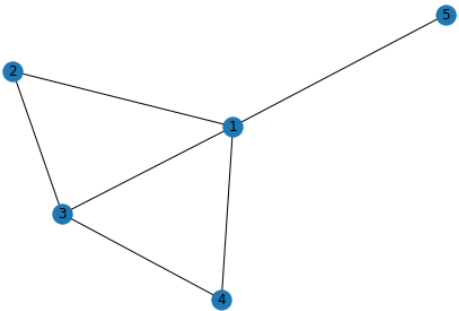
14

```
In [24]: # drawing in spectral layout
nx.draw_spectral(G, with_labels = True)
```



15

```
In [25]: # drawing in spring layout
nx.draw_spring(G, with_labels = True)
```



16

```
In [26]: # drawing in shell layout
nx.draw_shell(G, with_labels = True)
```

