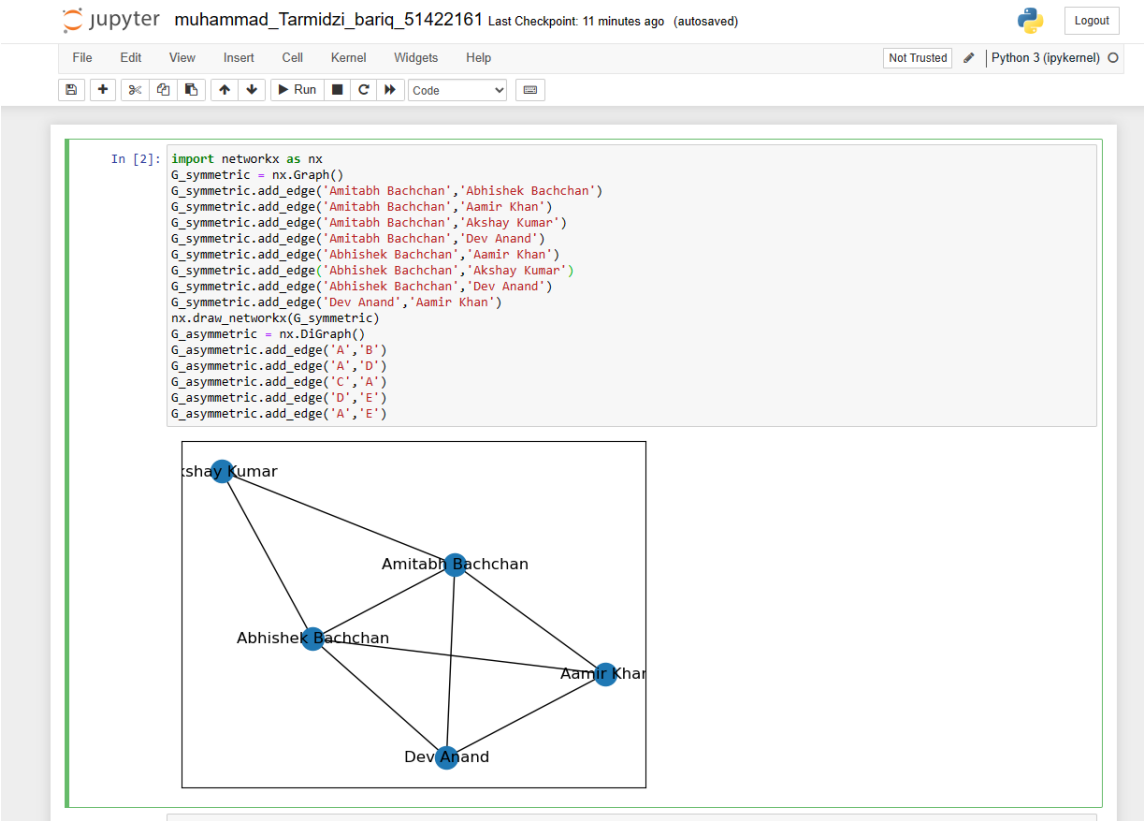
TUGAS PRAKTIKUM PERTEMUAN 6

NAMA : MUHAMMAD TARMIDZI BARIQ
KELAS : 1IA13
NPM : 51422161

1



Pada kode pemrograman tersebut, pertama-tama kita mengimport library NetworkX untuk mengakses algoritma dan fungsi yang berkaitan dengan graf dan jaringan.

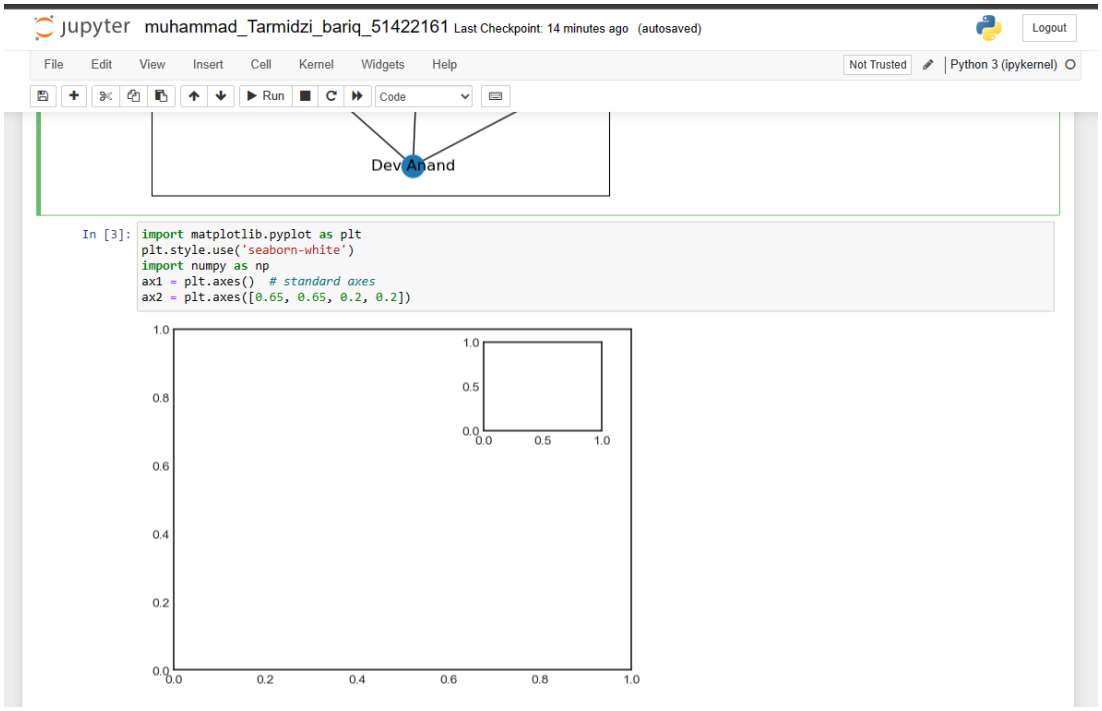
Kemudian kita membuat dua graf yang berbeda: G_symmetric dan G_asymmetric. Graf G_symmetric merupakan graf tak-berarah (undirected) yang berisi beberapa simpul (node) yang merepresentasikan nama-nama aktor Bollywood dan beberapa sisi (edge) yang merepresentasikan hubungan antara aktor-aktor tersebut. Graf ini didefinisikan dengan menggunakan fungsi nx.Graph() dan setiap sisi ditambahkan menggunakan fungsi add_edge().

Selanjutnya, graf G_symmetric digambarkan dengan menggunakan fungsi nx.draw_networkx().

Sedangkan, graf G_asymmetric merupakan graf berarah (directed) yang juga berisi beberapa simpul dan sisi yang merepresentasikan hubungan antar simpul. Graf ini didefinisikan dengan menggunakan fungsi nx.DiGraph() dan setiap sisi ditambahkan menggunakan fungsi add_edge().

Dalam contoh ini, simpul (node) direpresentasikan dengan karakter huruf dan sisi (edge) dihubungkan antara dua simpul dengan menggunakan fungsi add_edge(). Pada graf ini terdapat simpul A, B, C, D, dan E, dan hubungan antar simpul direpresentasikan oleh sisi-sisi yang menghubungkan simpul-simpul tersebut.

2

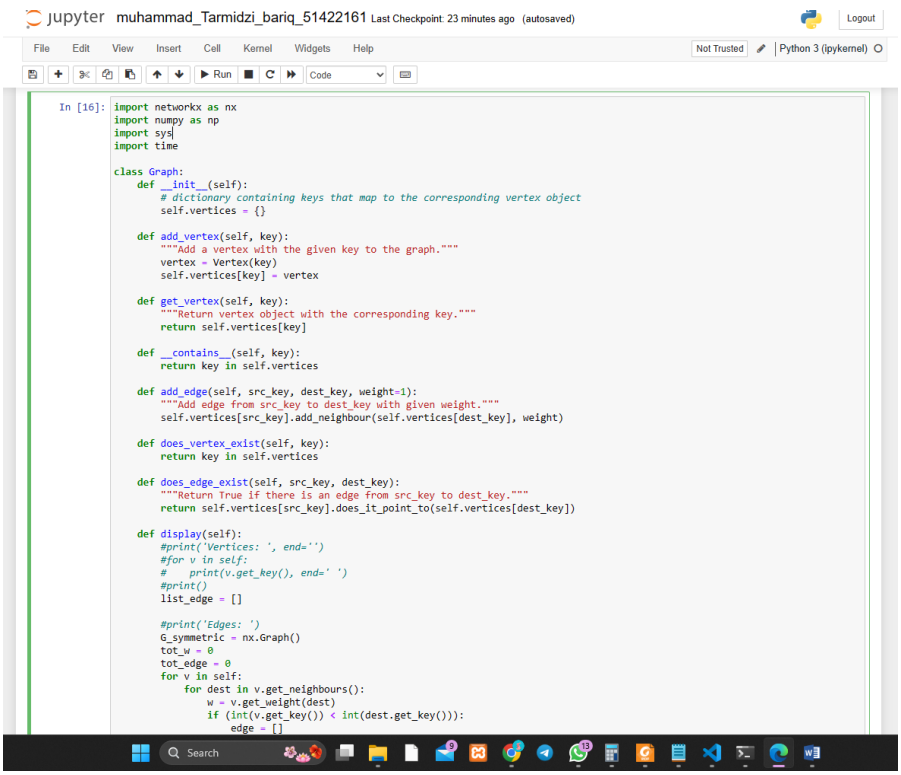


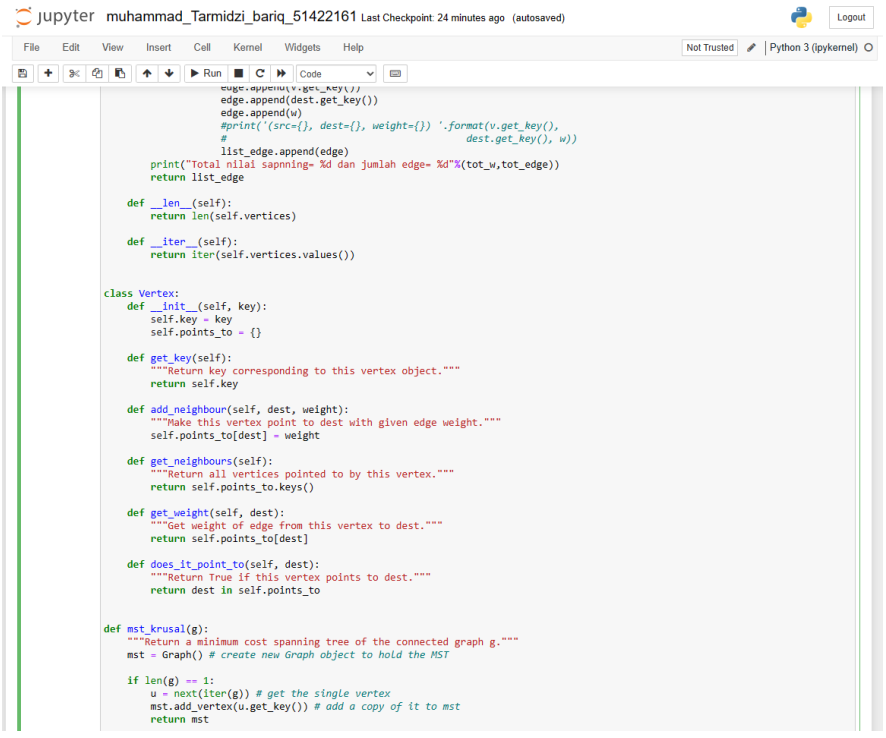
Pada kode pemrograman tersebut, pertama-tama kita mengimport library Matplotlib untuk mengakses algoritma dan fungsi yang berkaitan dengan visualisasi data.

Kemudian, kita mengatur gaya plot yang akan digunakan dengan menggunakan fungsi plt.style.use(). Pada contoh ini, gaya plot yang dipilih adalah 'seaborn-white'.

Selanjutnya, kita membuat dua buah objek axes untuk menampilkan dua plot yang berbeda. Objek pertama dibuat dengan menggunakan fungsi plt.axes() tanpa argumen, sehingga objek tersebut akan berukuran standar. Objek kedua dibuat dengan menggunakan fungsi plt.axes() dengan argumen berupa daftar [0.65, 0.65, 0.2, 0.2], yang menentukan posisi dan ukuran objek tersebut dalam koordinat gambar (0.65, 0.65) dan ukuran (0.2, 0.2).

3.

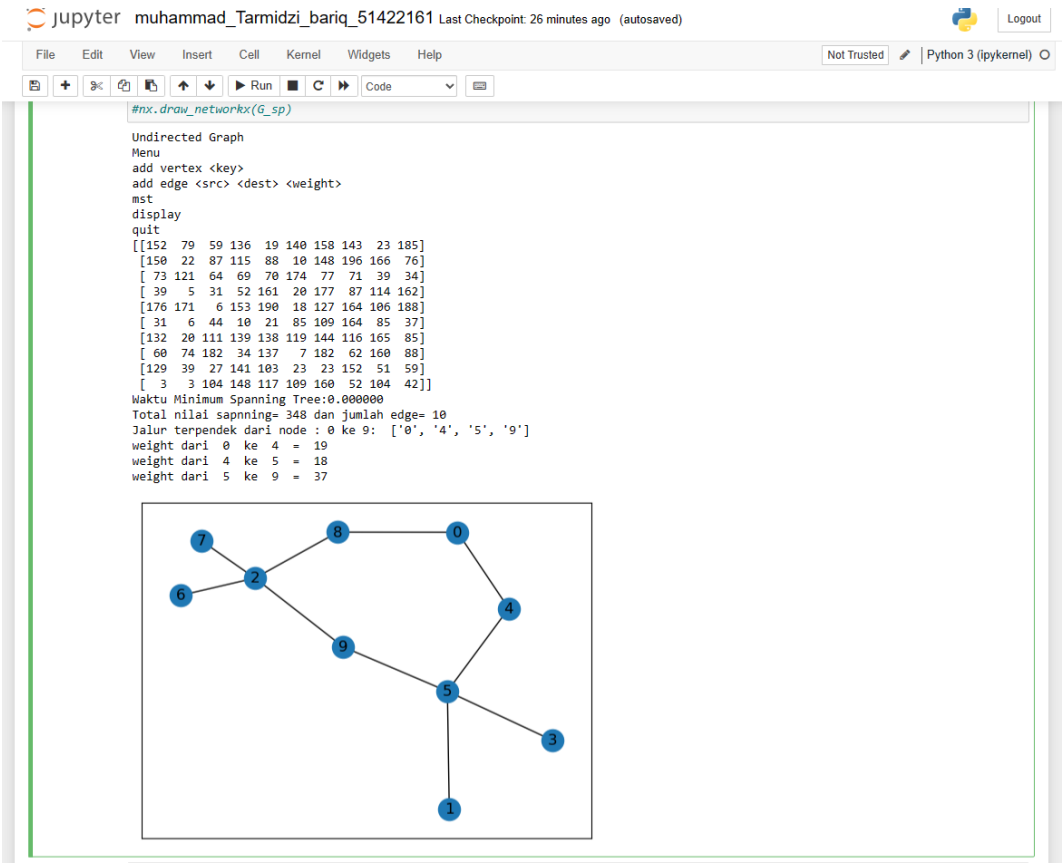

jupyter muhammad_Tarmidzi_bariq_51422161 Last Checkpoint: 23 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [16]:
import networkx as nx
import numpy as np
import sys
import time
class Graph:
def __init__(self):
# dictionary containing keys that map to the corresponding vertex object
self.vertices = {}
def add_vertex(self, key):
"""Add a vertex with the given key to the graph."""
vertex = Vertex(key)
self.vertices[key] = vertex
def get_vertex(self, key):
"""Return vertex object with the corresponding key."""
return self.vertices[key]
def __contains__(self, key):
return key in self.vertices
def add_edge(self, src_key, dest_key, weight=1):
"""Add edge from src_key to dest_key with given weight."""
self.vertices[src_key].add_neighbour(self.vertices[dest_key], weight)
def does_vertex_exist(self, key):
return key in self.vertices
def does_edge_exist(self, src_key, dest_key):
"""Return True if there is an edge from src_key to dest_key."""
return self.vertices[src_key].does_it_point_to(self.vertices[dest_key])
def display(self):
#print('Vertices: ', end='')
#for v in self:
print(v.get_key(), end=' ')
#print()
list_edge = []
#print('Edges: ')
G_symmetric = nx.Graph()
tot_w = 0
tot_edge = 0
for v in self:
for dest in v.get_neighbours():
w = v.get_weight(dest)
if (int(v.get_key()) < int(dest.get_key())):
edge = []
Search


jupyter muhammad_Tarmidzi_bariq_51422161 Last Checkpoint: 24 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
edge.append(v.get_key())
edge.append(dest.get_key())
edge.append(w)
#print('(src={}, dest={}, weight={}) '.format(v.get_key(),
# dest.get_key(), w))
list_edge.append(edge)
print("Total nilai sapnning= %d dan jumlah edge= %d"%(tot_w,tot_edge))
return list_edge
def __len__(self):
return len(self.vertices)
def __iter__(self):
return iter(self.vertices.values())
class Vertex:
def __init__(self, key):
self.key = key
self.points_to = {}
def get_key(self):
"""Return key corresponding to this vertex object."""
return self.key
def add_neighbour(self, dest, weight):
"""Make this vertex point to dest with given edge weight."""
self.points_to[dest] = weight
def get_neighbours(self):
"""Return all vertices pointed to by this vertex."""
return self.points_to.keys()
def get_weight(self, dest):
"""Get weight of edge from this vertex to dest."""
return self.points_to[dest]
def does_it_point_to(self, dest):
"""Return True if this vertex points to dest."""
return dest in self.points_to
def mst_krusal(g):
"""Return a minimum cost spanning tree of the connected graph g."""
mst = Graph() # create new Graph object to hold the MST
if len(g) == 1:
u = next(iter(g)) # get the single vertex
mst.add_vertex(u.get_key()) # add a copy of it to mst
return mst

```
    edge_index = 0

    # Loop until mst has the same number of vertices as g
    while len(mst) < len(g):
        u, v = edges[edge_index]
        edge_index += 1

        # if adding edge (u, v) will not form a cycle
        if component[u] != component[v]:

            # add to mst
            if not mst.does_vertex_exist(u.get_key()):
                mst.add_vertex(u.get_key())
            if not mst.does_vertex_exist(v.get_key()):
                mst.add_vertex(v.get_key())
            mst.add_edge(u.get_key(), v.get_key(), u.get_weight(v))
            mst.add_edge(v.get_key(), u.get_key(), u.get_weight(v))

            # merge components of u and v
            for w in g:
                if component[w] == component[v]:
                    component[w] = component[u]

    return mst


g = Graph()
print('Undirected Graph')
print('Menu')
print('add vertex <key>')
print('add edge <src> <dest> <weight>')
print('mst')
print('display')
print('quit')


N = 10

for i in range (0,N):
     g.add_vertex(str(i))

a = np.random.randint(2,200,(N,N), dtype=int)
print(a)

for i in range (0,N):
     for j in range (0,N):
         #if (i < j) :
             g.add_edge(str(i),str(j),a[i,j])

start = time.time()
```

```
for i in range (0,N):
     g.add_vertex(str(i))

a = np.random.randint(2,200,(N,N), dtype=int)
print(a)

for i in range (0,N):
     for j in range (0,N):
         #if (i < j) :
             g.add_edge(str(i),str(j),a[i,j])

start = time.time()
mst = mst_krusal(g)
print('Waktu Minimum Spanning Tree:%2f'%(time.time()-start))
l_edge = mst.display()
G_sym = nx.Graph()
for e in l_edge:
    G_sym.add_edge(e[0],e[1],weight=int(e[2]))
nx.draw_networkx(G_sym)

sh_path = nx.shortest_path(G_sym,'0', str(N-1))
G_sp = nx.Graph()
print("Jalur terpendek dari node : 0 ke 9: ", sh_path)

for i in range (len(sh_path)-1):
    v1 = g.get_vertex((sh_path[i]))
    v2 = g.get_vertex((sh_path[i+1]))
    print("weight dari ",v1.get_key()," ke ",v2.get_key()," = ",v1.get_weight(v2))
    G_sp.add_edge(v1,v2,weight=v1.get_weight(v2))

#nx.draw_networkx(G_sp)
```

```
Undirected Graph
Menu
add vertex <key>
```



Kode tersebut merupakan implementasi dari algoritma Kruskal untuk menemukan pohon rentang minimum dari graf tak berarah dan terhubung. Grafik direpresentasikan menggunakan daftar adjacency, dan kode menggunakan paket networkx untuk menampilkan grafik dan pohon rentang minimum. Kelas Grafik mewakili grafik, dan kelas Vertex mewakili sebuah simpul dalam grafik. Kelas Grafik berisi kamus yang memetakan kunci ke objek Vertex, dan menyediakan metode untuk menambahkan simpul dan tepi ke grafik, memeriksa apakah ada simpul atau tepi, dan iterasi di atas simpul. Fungsi mst_krusal mengambil objek Grafik sebagai input dan mengembalikan objek Grafik baru yang mewakili pohon rentang minimum dari grafik input. Fungsi pertama-tama membuat objek grafik kosong untuk menampung pohon rentang minimum. Jika graf masukan hanya berisi satu simpul, fungsi menambahkan salinan simpul ke pohon rentang minimum dan mengembalikannya. Jika tidak, fungsi membuat daftar sisi dalam graf masukan, mengurutkan sisi berdasarkan bobot, dan menginisialisasi setiap simpul untuk berada dalam komponen terhubungnya sendiri. Fungsi tersebut kemudian mengulang tepi, menambahkan setiap tepi ke pohon rentang minimum jika tidak membentuk siklus dan menggabungkan komponen yang terhubung dari titik akhir tepi. Metode tampilan kelas Grafik mencetak simpul dan tepi grafik dan mengembalikan daftar tepi yang dapat digunakan untuk membuat grafik networkx untuk ditampilkan. Fungsi nx.Graph digunakan untuk membuat grafik tak terarah baru, dan tepi dalam daftar ditambahkan ke grafik menggunakan metode

add_weighted_edges_from. Terakhir, fungsi draw_networkx digunakan untuk menampilkan grafik.