535230080-praktikum-06

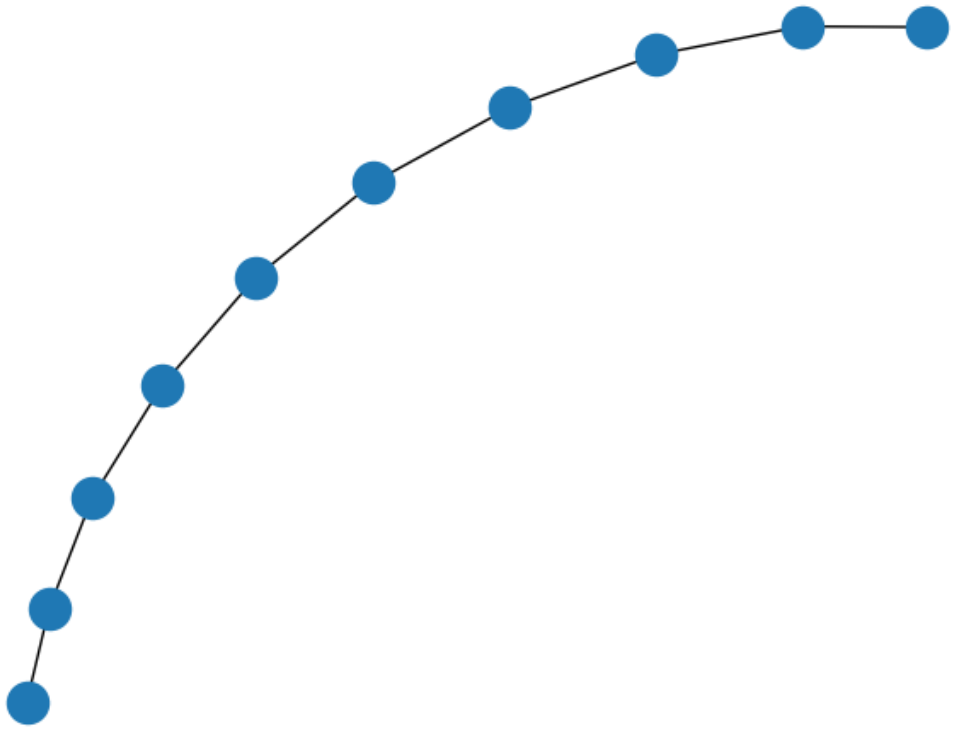
April 5, 2024

[399]:
```
#Georgia Sugisandhea_535230080_Kelas C
import networkx as nx #Mengimport library networkx sebagai variable nx
import matplotlib.pyplot as plt #Menginmport library matplotlib.pyplot sebagai␣
 ↪variable plt
```

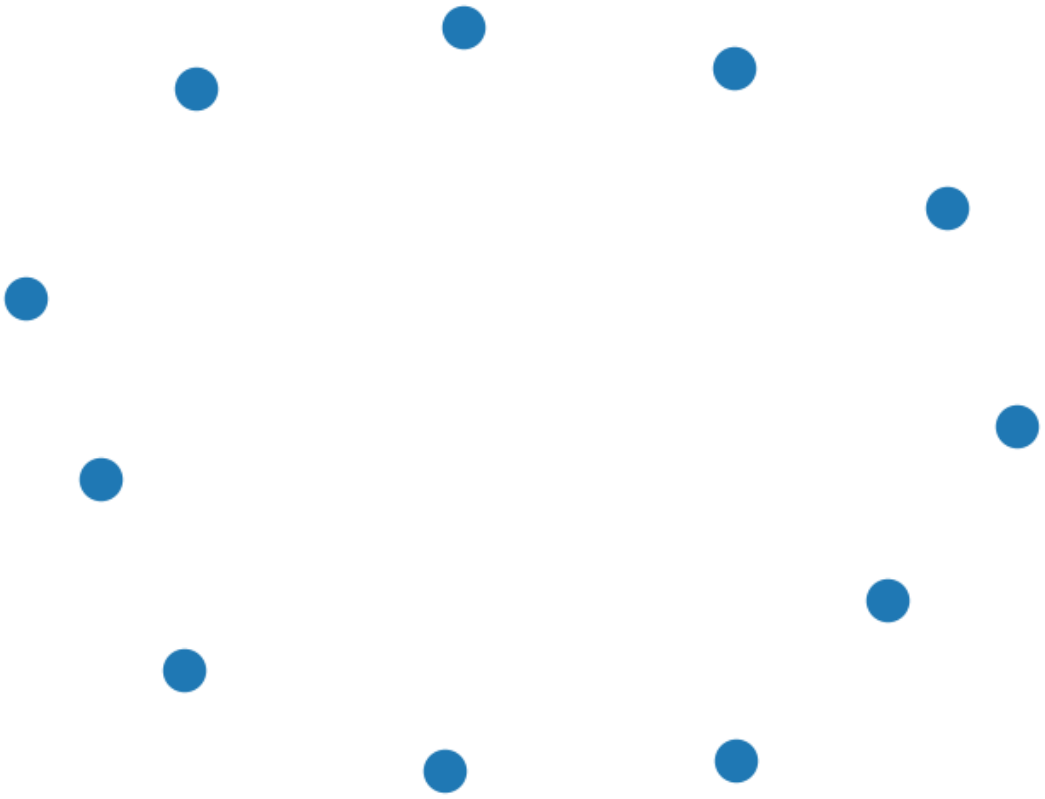
[400]:
```
#Membuat graph
G = nx.Graph()

#Menambahkan 1 node dengan label 1
G.add_node(1)

#Membuat grafik G dengan nx
nx.draw(G)

#Menampilkan hasil grafik G yang sudah dibuat
plt.show();
```

[401]:
```
#Menambahkan 2 node kedalam G dengan label 2 dan 3
G.add_nodes_from([2,3])

#Menggambar kembali grafik G
nx.draw(G)

#Menampilkan hasil grafik G yang sudah dibuat
plt.show()
```

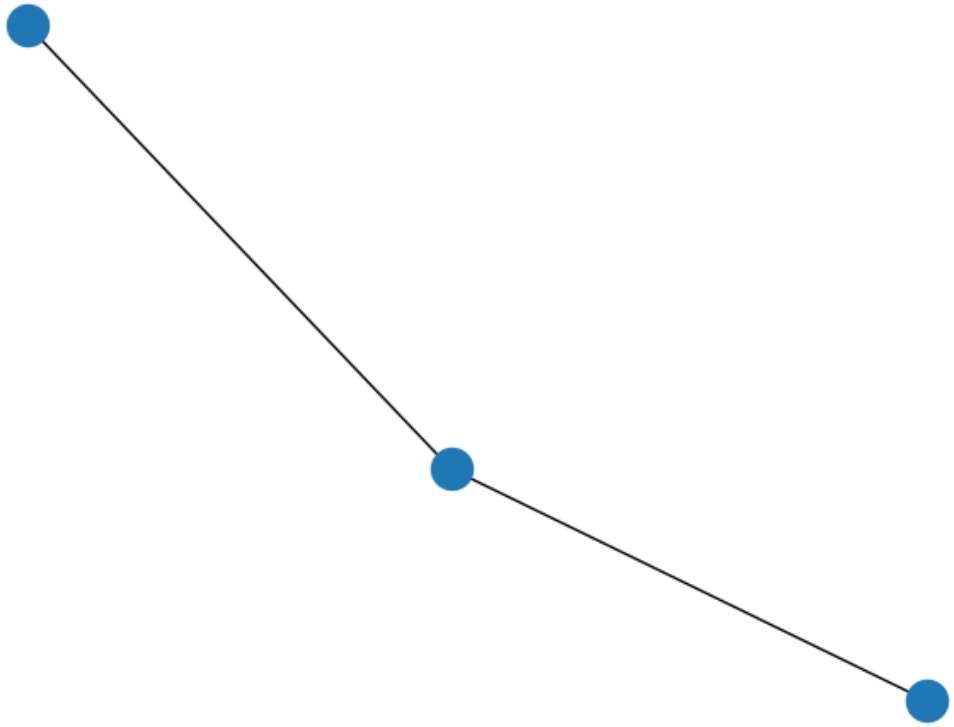
[402]:
```
#Menambahkan 10 node dengan label 0 sampai 9 yang saling terhubung di variable H
H = nx.path_graph(10)

#Menggambar hasil grafik H
nx.draw(H)

#Menampilkan hasil grafik H yang telah dibuat sebelumnya
plt.show()
```



[403]:
```
#Menambahkan anggota nodes H
G.add_nodes_from(H)

#Menambahkan nodes H, namun mereka tidak memasukkan node dengan label yang␣
 ↪sama, maka skip duplicate
G.add_node(H)

#Menggambar grafik dari G yang sudah kita tambahkan node dengan nx
nx.draw(G)

#Menampilkan grafik yang telah kita buat
plt.show()
```



[404]:
```
#Menghapus semua node dan edge yang ada di graph G
G.clear()

#Menambahkan node dan edge (penghubung antara node 1 dan 2)
G.add_edge(1,2)

#Membuat variable e yang berisi 2 dan 3
e = (2,3)

#Menambahkan node dan edge dari variable e yang menghubungkan antara node 2 dan␣
↪3
G.add_edge(*e)

#Menggambar grafik G yang telah kita buat dan isi node dan edge
nx.draw(G)

#Menampilkan grafik yang telah dibuat
plt.show()
```



[405]:
```
#Menghapus semua node dan edge yang ada di graph G
G.clear()

#Menambahkan node dan edges yang menghubungkan 1 dan 2 kemudian 1 dan 3
G.add_edges_from([(1,2), (1,3)])

#Menambahkan node dengan label 1
G.add_node(1)

#Menambahkan penghubung antara 1 dan 2
G.add_edge(1,2)

#Menambahkan node dengan label spam
G.add_node("spam")

#Menambahkan anggota node dari spam, yaitu s p a m
G.add_nodes_from("spam")

#Mengubungkan node 3 dan node m
G.add_edge(3,'m')
```

[406]:
```
#Menampilkan jumlah node yang ada
G.number_of_nodes()
```

[406]: 8

[407]:
```
#menampilkan jumlah edges(penghubung) yang ada
G.number_of_edges()
```

[407]: 3

[408]:
```
#Membuat Directed Graph baru dengan variable DG
DG = nx.DiGraph()

#Menambahkan edges (penghubung) antara node node yang dimasukkan dalam ()
DG.add_edge(2,1)
DG.add_edge(1,3)
DG.add_edge(2,4)
DG.add_edge(1,2)

#Memastikan 2 memiliki successors 1 dan 4, jika tidak maka error
assert list(DG.successors(2)) == [1,4]

#Memastikan masing masing node yang dimasukkan dalam () saling terhubung, jika␣
 ↪tidak maka error
assert list(DG.edges)==[(2,1), (2,4), (1,3), (1,2)]
```

[409]:
```
#Menampilkan node node yang ada dalam variable G
list(G.nodes)
```

[409]: [1, 2, 3, 'spam', 's', 'p', 'a', 'm']

[410]:
```
#Menampilkan edges(node yang terhubung) yang ada di dalam variable G
list(G.edges)
```

[410]: [(1, 2), (1, 3), (3, 'm')]

[411]:
```
#list node yang dihubungkan dengan node 1 pada graph G
list(G.adj[1])
```

[411]: [2, 3]

[412]:
```
#Menampilkan node node yang terhubung kepada node 2 dan m
G.edges([2,'m'])
```

[412]: EdgeDataView([(2, 1), ('m', 3)])

[413]:
```
#Menampilkan degree dari node node yang dimasukkan pada graph G
G.degree([2,3])
```

[413]:
```
DegreeView({2: 1, 3: 2})
```

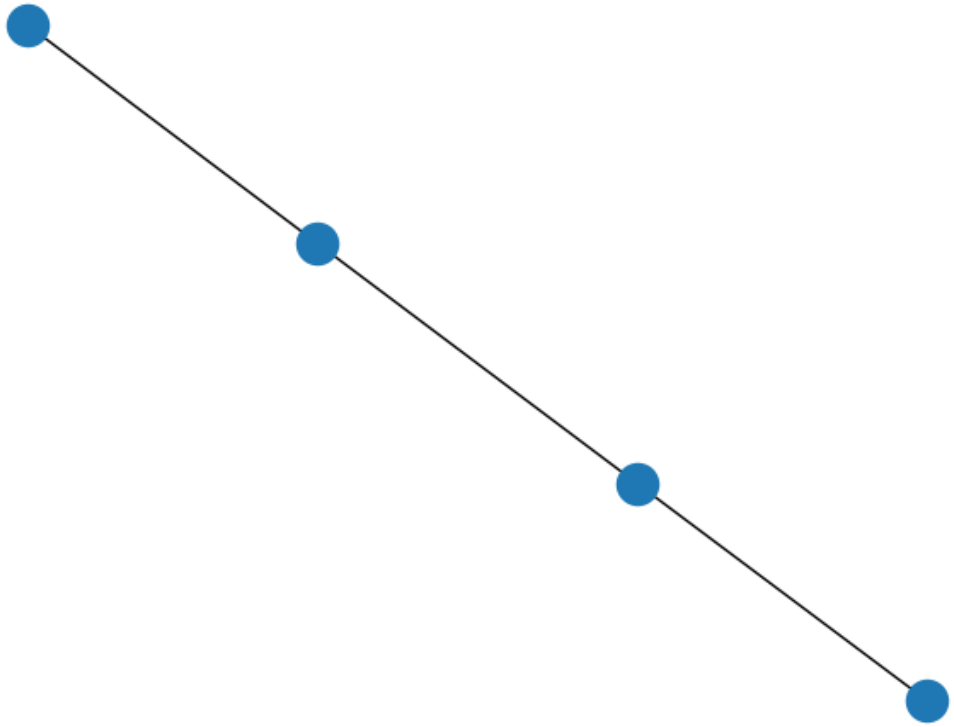
[414]:
```
#Menghapus node 2 dari G
G.remove_node(2)
#Menghapus semua anggota node dari spam (s, p, a, m) dari G
G.remove_nodes_from("spam")
#Menghapus edge(hubungan) antara 1 dan 3
G.remove_edge(1,3)
#Menunjukkan anggota anggota node yang ada di G
list(G.nodes)
```

[414]:
```
[1, 3, 'spam']
```

[415]:
```
#Menambahkan node dan edge (penghubung antara node 1 dan 2)
G.add_edge(1,2)

#Membuat Directed Graph dari G kemudia dimasukkan ke dalam H
H = nx.DiGraph(G)

#Menampilkan edges(hubungan) yang ada di list H
list(H.edges())
```

[415]:
```
[(1, 2), (2, 1)]
```

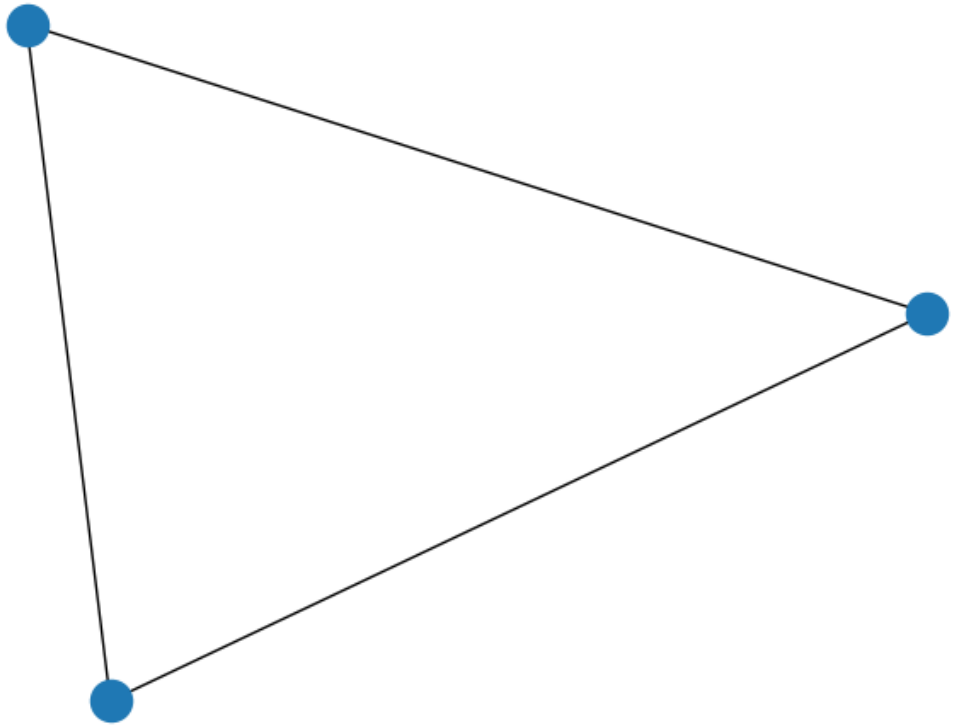
[416]:
```
#Membuat list array 2 dimensi dengan nama edgeList
edgelist=[(0,1), (1,2), (2,3)]
#Membuat Graph dengan nx dengan variable H, dengan edges yang sudah kita buat␣
 ↪dengan variable edgeList
H = nx.Graph(edgelist)
#Menggambar graph H yang telah kita isi sebelumnya
nx.draw(H)
#Menampilkan hasil grafik yang telah kita buat
plt.show()
```



[417]:
```
#Membuat list dimana n:(x,y) berarti node n berhubungan dengan node x dan y
adjacency_dict={0: (1,2), 1: (0,2), 2: (0,1)}
#Membuat graph yang dimasukkan ke variable H dengan list yang sudah kita buat␣
 ↪sebelumnya
H = nx.Graph(adjacency_dict)
#Menampilkan edges(hubungan) yang ada pada variable H
list(H.edges())
```

[417]:
```
[(0, 1), (0, 2), (1, 2)]
```

[418]:
```
#Menggambar graph H yang sudah kita buat
nx.draw(H)
#Menampilkan gambar graph yang telah kita buat
plt.show()
```



[419]:
```
#Membuat graph dengan node 1 dan 2 dan atribut warna kuning
G = nx.Graph([(1,2,{"color": "yellow"})])
```

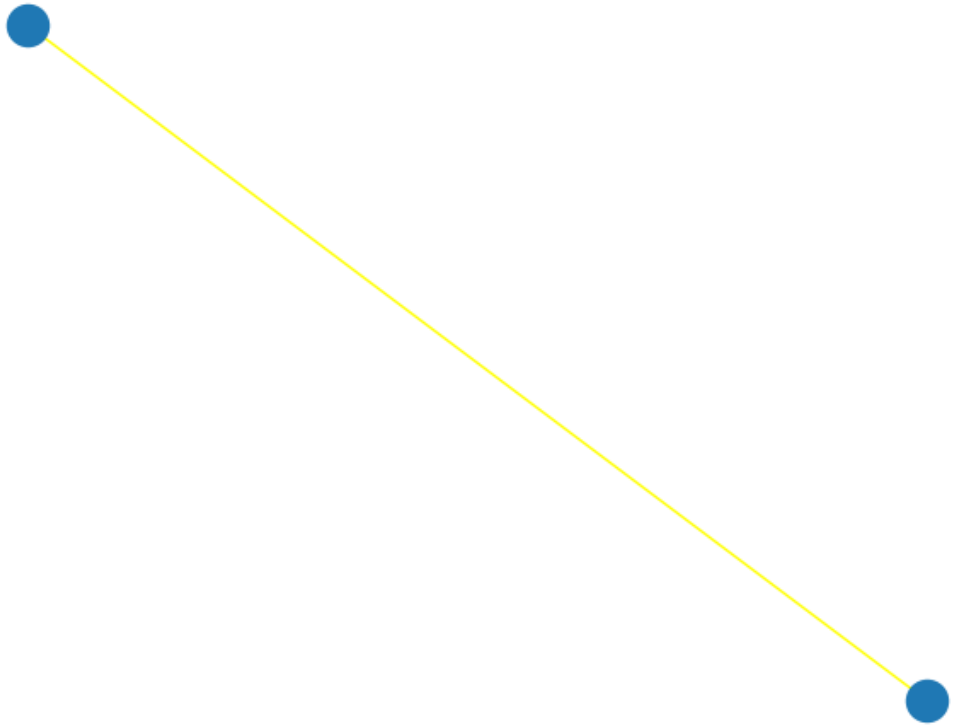
[420]:
```
#Mengakses tetangga dari node 1
G[1]
```

[420]:
```
AtlasView({2: {'color': 'yellow'}})
```

[421]:
```
#Mengakses atribut node 1 dan 2
G[1][2]
```

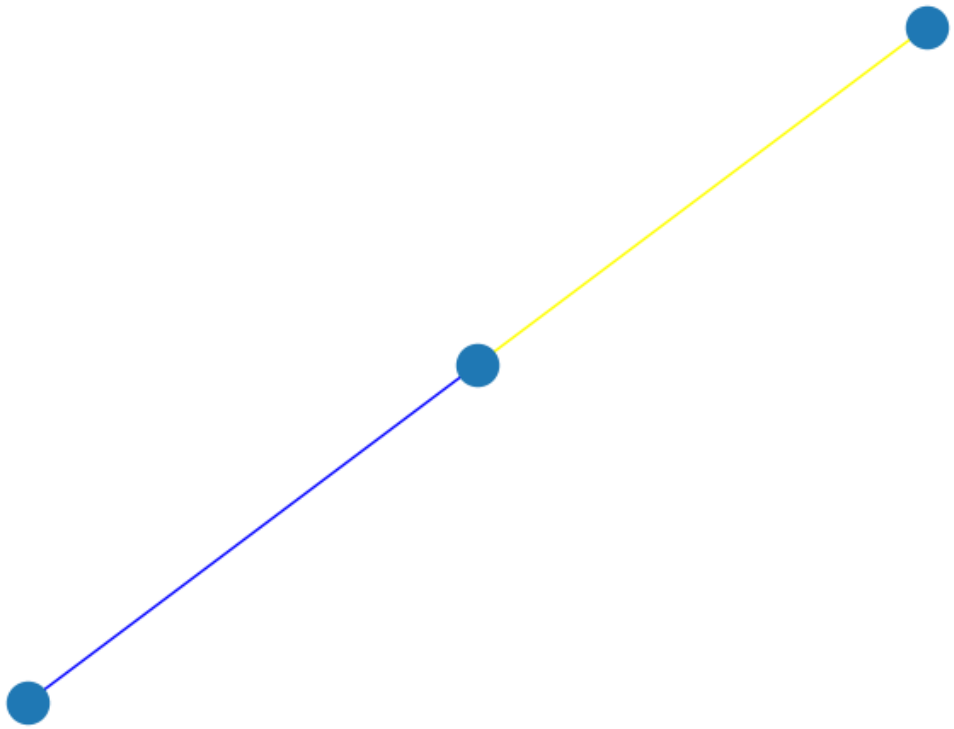
[421]:
```
{'color': 'yellow'}
```

[422]:
```
#Menggambar grafik G dengan warna yang telah ditentukan sebelumnya
nx.draw(G, edge_color=[G.edges[edge]['color'] for edge in G.edges()])
#Menampilkan hasil grafik yang telah kita buat sebelumnya
plt.show()
```



[423]:
```
#Menambah edge(hubungan) antara node 1 dan 3
G.add_edge(1,3)
#Menambah atribut hubungan antara 1 dan 3 sebagai warna biru
G[1][3]['color'] = "blue"
```

[424]:
```
#Kembali membuat dan menampilkan graph beserta warna atribut yang telah dibuat
nx.draw(G, edge_color=[G.edges[edge]['color'] for edge in G.edges()])
plt.show()
```



[425]:
```
#Membuat warna hubungan(edges) antara node 1 dan 2 bewarna merah
G.edges[1,2]['color']="red"
```

[426]:
```
#Menampilkan atribut hubungan node 1 dan 2
G.edges[1,2]
```

[426]:
```
{'color': 'red'}
```

[427]:
```
#Menampilkan edges(hubungan) yang ada di variable FG yang weightnya lebih kecil␣
 ↪dari 0.5 dan jumlah angka belakang koma sebanyak 3
FG = nx.Graph()
FG.add_weighted_edges_from([(1,2,0.125), (1,3,0.75), (2,4,1.2), (3,4,0.375)])
for n, nbrs in FG.adj.items():
    for nbr, eattr in nbrs.items():
        wt = eattr['weight']
        if wt < 0.5: print(f"({n}, {nbr}, {wt:.3})")

#Duplikat bolak balik karena mereka menampilkan hubungan satu arah dan␣
 ↪sebaliknya
```

```
(1, 2, 0.125)
```

```
(2, 1, 0.125)
(3, 4, 0.375)
(4, 3, 0.375)
```

[428]:
```
#Menampilkan node dan juga neighbors(nbrs) beserta weight masing masing hubungan
for n, nbrs in FG.adj.items():
    print("n: \n nbrs:{}" .format(n, nbrs))
    for nbr, eattr in nbrs.items():
        print("nbr: {}\n eattr: {}" .format(nbr, eattr))
```

```
n:
 nbrs:1
nbr: 2
 eattr: {'weight': 0.125}
nbr: 3
 eattr: {'weight': 0.75}
n:
 nbrs:2
nbr: 1
 eattr: {'weight': 0.125}
nbr: 4
 eattr: {'weight': 1.2}
n:
 nbrs:3
nbr: 1
 eattr: {'weight': 0.75}
nbr: 4
 eattr: {'weight': 0.375}
n:
 nbrs:4
nbr: 2
 eattr: {'weight': 1.2}
nbr: 3
 eattr: {'weight': 0.375}
```

[429]:
```
#Menampilkan edges(hubungan) yang ada di variable FG yang weightnya lebih kecil␣
 ↪dari 0.5 dan jumlah angka belakang koma sebanyak 3
for (u,v,wt) in FG.edges.data('weight'):
    if wt<0.5:
        print(f"({u}, {v}, {wt:.3})")
#Kali ini ditampilkan tidak duplikat bolak balik
```

```
(1, 2, 0.125)
(3, 4, 0.375)
```

[430]:
```
#Membuat grafik G dan juga mengisi node day dengan value Friday
G = nx.Graph(day="Friday")
#dan menampilkan grafik G yang telah dibuat
```

```
G.graph
```

[430]:
```
{'day': 'Friday'}
```

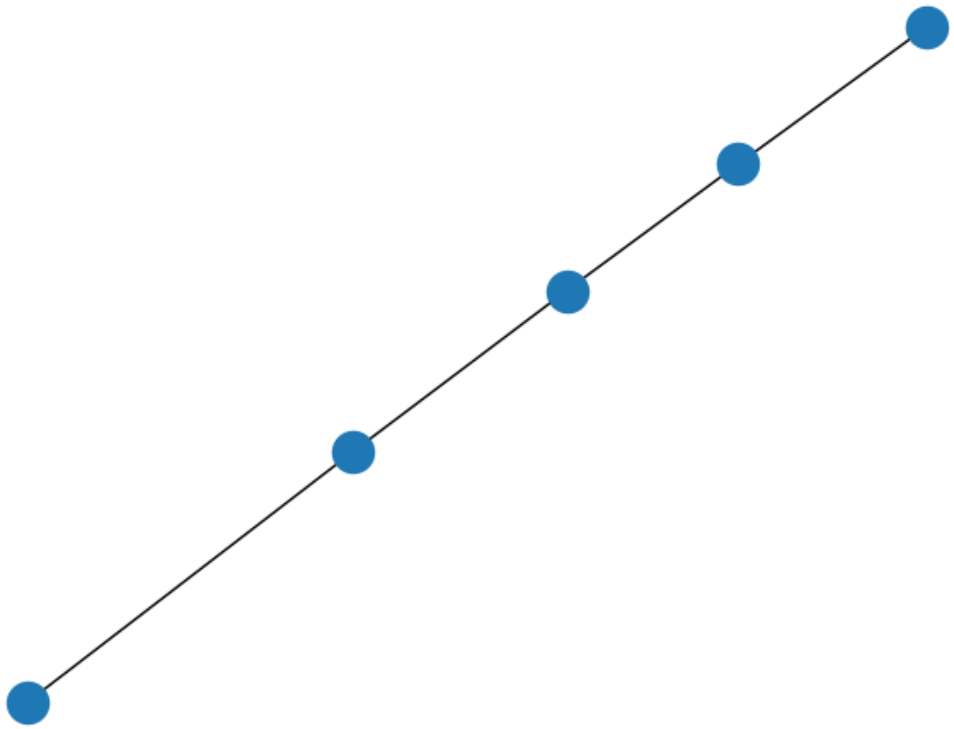
[431]:
```
#Mengganti value day di grafik G menjadi Monday
G.graph['day'] = "Monday"
#dan menampilkan hasilnya
G.graph
```

[431]:
```
{'day': 'Monday'}
```

[432]:
```
##Menambahkan node pada variable G masing masing dengan time dan valuenya
G.add_node(1, time='5pm')
G.add_node(3, time='2pm')
#Mengganti value time di node 3 dengan 2pm
G.add_nodes_from([3], time='2pm')
#Menambahkan room pada node 1 dengan value 714
G.nodes[1]
G.nodes[1]['room'] = 714
#Menampilkan node dan juga data yang disimpan di variable G
G.nodes.data()
```

[432]:
```
NodeDataView({1: {'time': '5pm', 'room': 714}, 3: {'time': '2pm'}})
```

[433]:
```
#Menghapus semua node dan edges yang ada di variable G
G.clear()
#Menambahkan hubungan antara node 1 dan 2 beserta weightnya
G.add_edge(1,2, weight=4.7)
#Menambahkan hubungan antara node 3 dan 4, 4 dan 5 dan dengan color red
G.add_edges_from([(3,4), (4,5)], color='red')
#Menambahkan atribut hubungan node 1 dan 2 menjadi warna biru, dan menambah␣
 ↪hubungan antara 2 dan 3 dengan weight 8
G.add_edges_from([(1,2,{'color': 'blue'}), (2,3,{'weight':8})])
#Menambahkan weight node 1 dan 2 menjadi 4.7
G[1][2]['weight']=4.7
#Menambahkan weight hubungan dantara node 3 dan 4 menjadi 4.2
G.edges[3,4]['weight'] = 4.2
#Menggambar grafik yang telah dibuat dan diisi
nx.draw(G)
#Menampilkan hasil grafik yang dibuat
plt.show()
```

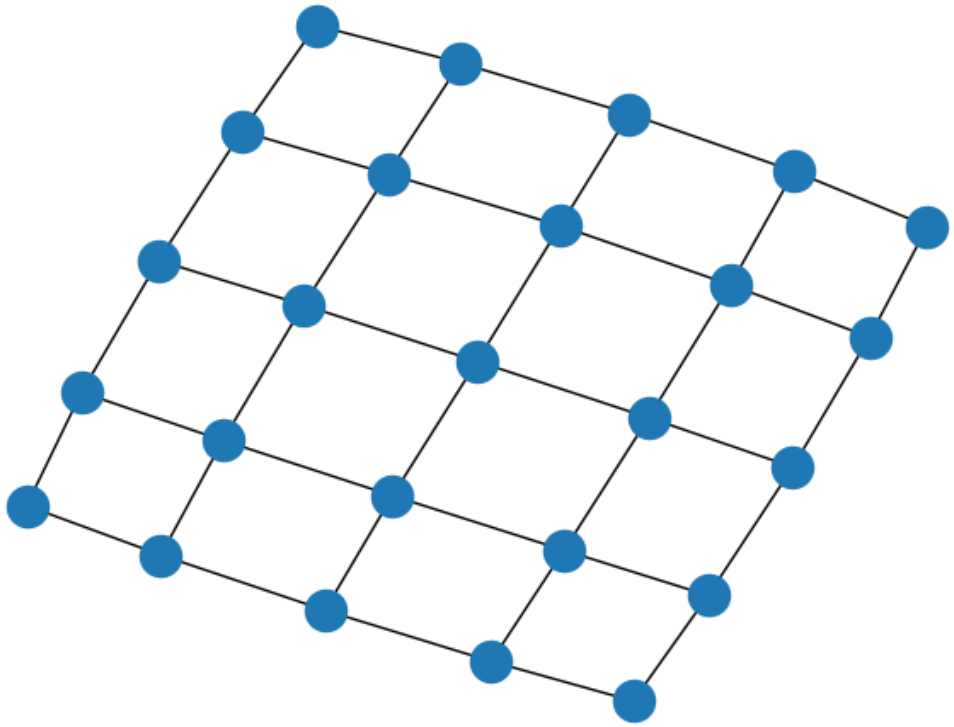


[434]:
```
#Membuat grafik variable G berupa grid 5x5
G = nx.grid_2d_graph(5,5) #5x5 grid
```

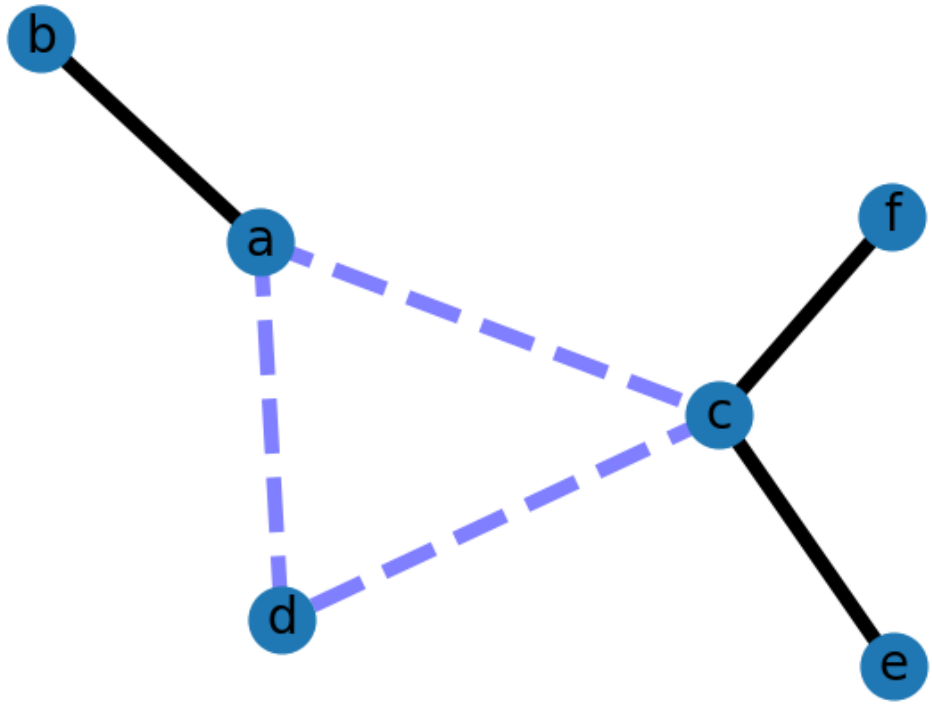
[435]:
```
#Menampilkan setiap node beserta neighborsnya
for line in nx.generate_adjlist(G):
    print(line)
#Membuat edgelist yang berisi grid edgelisyt yang ada di variable G dan␣
 ↪dipisahkan dekimiter :
nx.write_edgelist(G, path="grid.edgelist", delimiter=":")
#Membaca edgelist yang telah dibuat dan dimasukkan ke variable H
H = nx.read_edgelist(path="grid.edgelist", delimiter=":")

#Membuat grafik H dan memiliki layout berupa spring layout
pos = nx.spring_layout(H, seed=200)
#Menggambar grafik H beserta posisi layout yang telah dibuat
nx.draw(H, pos)
#Menampilkan hasil grafik yang telah dibuat
plt.show()
```

```
(0, 0) (1, 0) (0, 1)
(0, 1) (1, 1) (0, 2)
```

```
(0, 2) (1, 2) (0, 3)
(0, 3) (1, 3) (0, 4)
(0, 4) (1, 4)
(1, 0) (2, 0) (1, 1)
(1, 1) (2, 1) (1, 2)
(1, 2) (2, 2) (1, 3)
(1, 3) (2, 3) (1, 4)
(1, 4) (2, 4)
(2, 0) (3, 0) (2, 1)
(2, 1) (3, 1) (2, 2)
(2, 2) (3, 2) (2, 3)
(2, 3) (3, 3) (2, 4)
(2, 4) (3, 4)
(3, 0) (4, 0) (3, 1)
(3, 1) (4, 1) (3, 2)
(3, 2) (4, 2) (3, 3)
(3, 3) (4, 3) (3, 4)
(3, 4) (4, 4)
(4, 0) (4, 1)
(4, 1) (4, 2)
(4, 2) (4, 3)
(4, 3) (4, 4)
(4, 4)
```

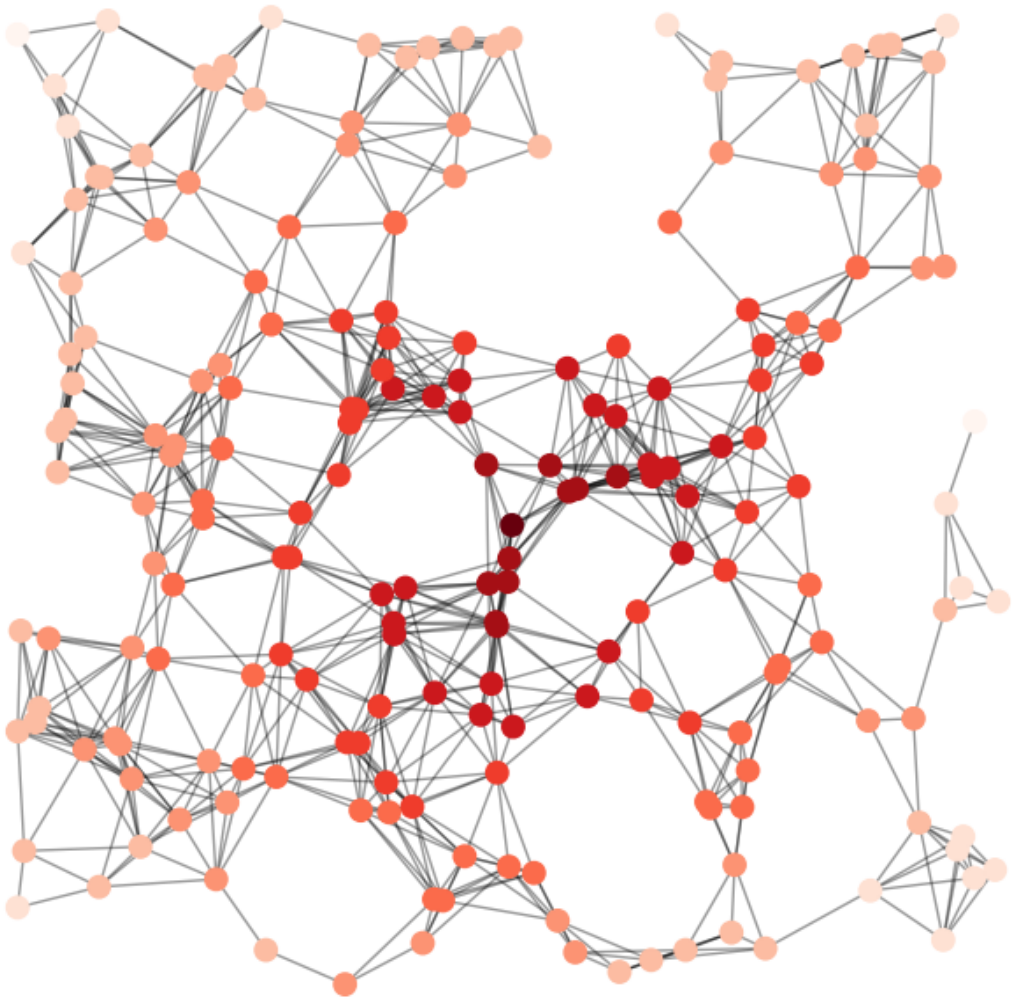


[436]:
```
#Membuat grafik yang berisi node dan edges
#dimana ketika edgesnya lebih besar dari 0.5, garisnya solid
#dan ketika edgesnya lebih kecil atau sama dengan 0.5, garisnya putus putus
#dan masing masing label dari node yang ada ditampilkan labelnya
G = nx.Graph()

G.add_edge("a", "b", weight=0.6)
G.add_edge("a", "c", weight=0.2)
G.add_edge("c", "d", weight=0.1)
G.add_edge("c", "e", weight=0.7)
G.add_edge("c", "f", weight=0.9)
G.add_edge("a", "d", weight=0.3)

elarge = [(u,v) for (u,v,d) in G.edges(data=True) if d["weight"]>0.5]
esmall = [(u,v) for (u,v,d) in G.edges(data=True) if d["weight"]<=0.5]

pos = nx.spring_layout(G, seed=7)

nx.draw_networkx_nodes(G, pos, node_size=700)

nx.draw_networkx_edges(G, pos, edgelist=elarge, width=6)
nx.draw_networkx_edges(
    G, pos, edgelist=esmall, width=6, alpha=0.5, edge_color="b", style="dashed"
)

nx.draw_networkx_labels(G, pos, font_size=20, font_family="sans-serif")

ax = plt.gca()
ax.margins(0.08)
plt.axis("off")
plt.tight_layout()
plt.show()
```

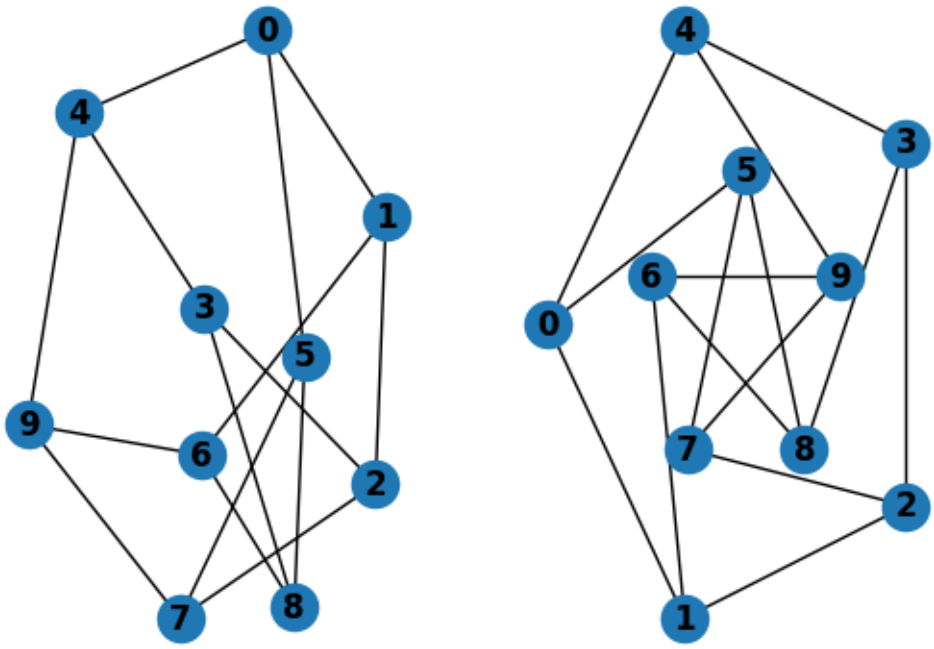


[437]:
```
#Membuat grafik dengan node geometri yang random
#Dimana ada 200 node yang masing masing terhubung dengan node node lain dengan␣
 ↪jarak yang tidak lebih dari 0.125
#Semakin jauh dari tengah grafik, node yang ditampilkan berwarna semakin␣
 ↪terang, sebaliknya, makin dekat makin gelap
#Dan membatasi grafik sumbu x dan y pada -0.05 dan 1.05
G = nx.random_geometric_graph(200, 0.125, seed=8696803)

pos = nx.get_node_attributes(G, "pos")

dmin = 1
ncenter = 0
for n in pos:
    x,y = pos[n]
    d = (x-0.5)**2 + (y-0.5)**2
    if d<dmin:
        ncenter=n
        dmin=d

p = dict(nx.single_source_shortest_path_length(G, ncenter))
```

```
plt.figure(figsize=(8,8))
nx.draw_networkx_edges(G, pos, alpha=0.4)
nx.draw_networkx_nodes(
    G,
    pos,
    nodelist=list(p.keys()),
    node_size=80,
    node_color=list(p.values()),
    cmap=plt.cm.Reds_r,
)

plt.xlim(-0.05, 1.05)
plt.ylim(-0.05, 1.05)
plt.axis("off")
plt.show()
```



[438]:
```
#Membuat 2 grafik petersen dimana ada 10 node dam 15 edges
#Kedua grafik ditampilkan dengan subplot 2 kolom 1 baris
#Menampilkan grafiknya beserta dengan label yang dibuat bold
#Grafik ke2 menggunakan shell layout dimana node 0 sampai 4 ada di luar dan␣
 ↪node 5 sampai 9 ada di dalam
G = nx.petersen_graph()
subax1 = plt.subplot(121)
nx.draw(G, with_labels=True, font_weight='bold')
subax2 = plt.subplot(122)
nx.draw_shell(G, nlist=[range(5,10), range(5)], with_labels=True,␣
 ↪font_weight='bold')
plt.show()
```



[439]:
```
#Mengimport library numpy sebagai variable np
import numpy as np

#Mengimport normalized_mutual_info_score dari library sklearn.metrics.cluster
from sklearn.metrics.cluster import normalized_mutual_info_score

#Mengimport library pandas sebagai variable pd
import pandas as pd
```

[440]:
```
#Membuat fungsi DataGraph untuk memasukkan data ke dalam sebuah array
#data yang dimasukkan adalah data yang memiliki upper triangular posisi␣
 ↪tersebut lebih besar dari threshold yang ditentukan

def DataGraph(dep):
    r1=dep.shape[0]
    c1=dep.shape[1]
    threshold = 0.8
    up = np.triu(dep, k=0)
    var = ["N1", "N2", "N3", "N4", "N5", "N6", "N7", "N8", "N9", "N10", "N11",␣
 ↪"N12", "N13", "N14", "N15", "N16", "N17", "N18", "N19", "N20", "N21", "N22",␣
 ↪"N23", "N24", "N25"]
    graph=[]
    for i in range(0,r1):
        for j in range(0,c1):
            if up[i][j]>threshold:
                x=var[i]
                y=var[j]
                temp=[x,y]
                graph.append(temp)
    return graph
```

[441]:
```
#Fungsi untuk menggambar grafik yang telah lengkap dengan atributnya, untuk␣
 ↪node, labels, dan edges, seperti layout, size, color, dkk
def draw_graph(graph, labels=None, graph_layout = 'shell',
               node_size=1600, node_color='blue', node_alpha=0.2,
               node_text_size=12,
               edge_color='blue', edge_alpha=0.4, edge_thickness=1,
               edge_text_pos=0.3, text_font='sans_serif'):
    G=nx.Graph()
    for edge in graph:
        G.add_edge(edge[0], edge[1])

    if graph_layout=='spring':
        graph_pos=nx.spring_layout(G)
    elif graph_layout=='spectral':
        graph_pos=nx.spectral_layout(G)
    elif graph_layout=='random':
        graph_pos=nx.random_layout(G)
    else:
        graph_pos=nx.shell_layout(G)

    pos=nx.fruchterman_reingold_layout(G)

    nx.draw_networkx_nodes(G, pos, node_size=node_size, alpha=node_alpha,␣
 ↪node_color=node_color)
```

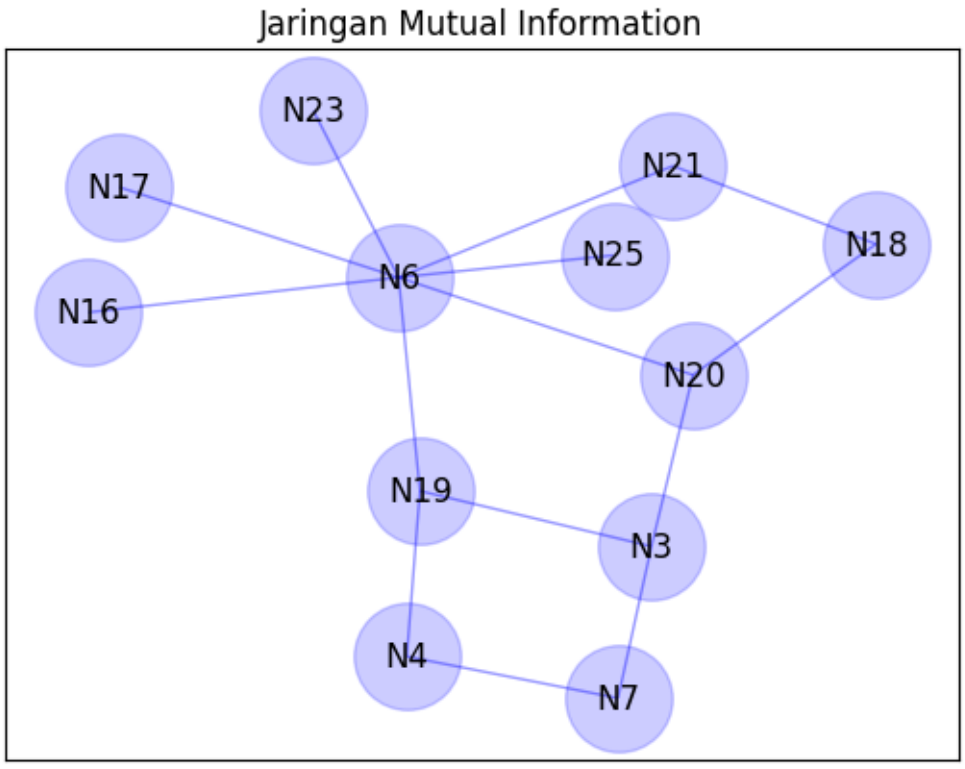
```
    nx.draw_networkx_edges(G, pos, width=edge_thickness, alpha=edge_alpha,␣
 ↪edge_color=edge_color)

    nx.draw_networkx_labels(G, pos, font_size=node_text_size,␣
 ↪font_family=text_font)

    if labels is None:
        labels = range(len(graph))

    plt.show()
```

[442]:
```
#Membaca file MIScore.xlsx yang ada di folder yang sama dengan file ipynb ini
#Dan membaca file tersebut di bagian Sheet1
#Membaca file menggunakan library pandas
file1 = 'MIScore.xlsx'
sheet1='Sheet1'

dep = pd.read_excel(file1, sheet1)
```

[443]:
```
#Mengelola data yang telah dibaca sebelumnya dengan fungsi yang telah kita buat␣
 ↪sebelumnya (DataGraph)
graph = DataGraph(dep)

#Memberi label kepada node node sebagai huruf yang diubah dari angka (chr())␣
 ↪dengan range 65 sampai 65+panjang graph
labels = map(chr, range(65, 65+len(graph)))

#Membuat judul untuk grafiknya
plt.title("Jaringan Mutual Information")

#Menampilkan hasil grafik tersebut
draw_graph(graph)
```

```
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
```

```
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
```



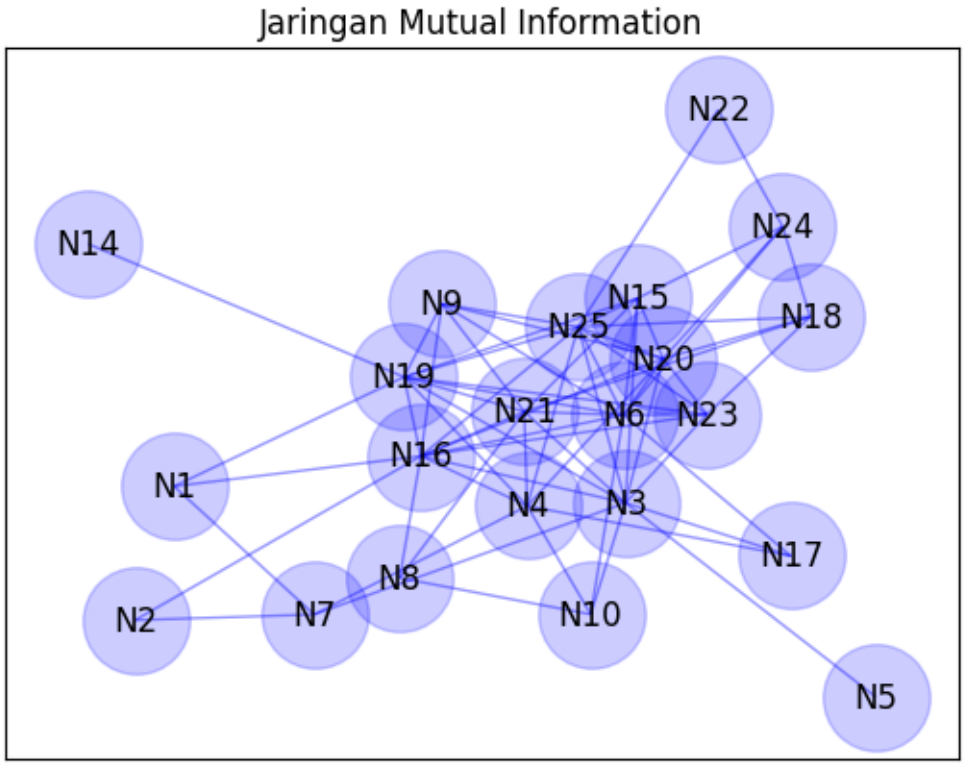
[444]:
```
#fungsi dan implementasi yang sama dengan yang sebelum ini, namun dengan␣
 ↪threshold 0.75 dan menghasilkan grafik yang jauh lebih kompleks
def DataGraph(dep):
    r1=dep.shape[0]
    c1=dep.shape[1]
    threshold = 0.75
    up = np.triu(dep, k=0)
    var = ["N1", "N2", "N3", "N4", "N5", "N6", "N7", "N8", "N9", "N10", "N11",␣
 ↪"N12", "N13", "N14", "N15", "N16", "N17", "N18", "N19", "N20", "N21", "N22",␣
 ↪"N23", "N24", "N25"]
    graph=[]
    for i in range(0,r1):
        for j in range(0,c1):
            if up[i][j]>threshold:
                x=var[i]
                y=var[j]
                temp=[x,y]
                graph.append(temp)
    return graph
```

[445]:
```
#implementasi yang sama dengan yang sebelum ini, namun fungsinya memiliki␣
 ↪threshold 0.75 dan menghasilkan grafik yang jauh lebih kompleks
graph = DataGraph(dep)

labels = map(chr, range(65, 65+len(graph)))

plt.title("Jaringan Mutual Information")
draw_graph(graph)
```

```
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
```

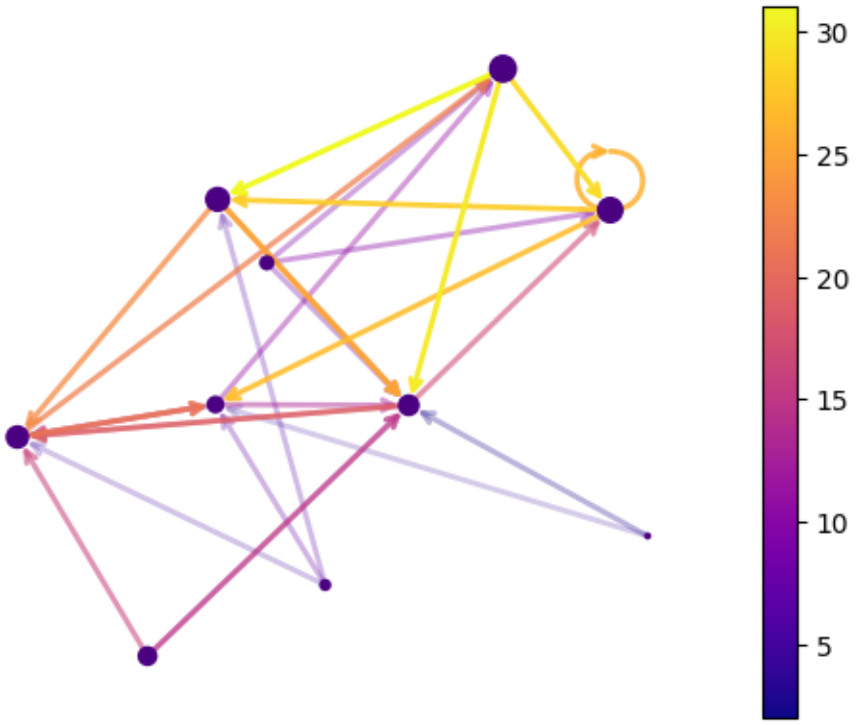
```
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
```

```
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
findfont: Font family 'sans_serif' not found.
```



[446]:
```
#Membuat grafik dengan node dan edges yang random, yang hasilnya warna warni␣
 ↪sesuai dengan array yang dibuat dengan range(2, jumlah node+2)
#Dimana hasilnya juga akan ada color bar yang menunjukkan range yang didapatkan␣
 ↪pada edge tersebut, dan node menggunakan warna indigo
#Grafik ini menggunakan layout spring
#Grafik ini menggunakan garis yang berarah juga, sehingga masing masing␣
 ↪hubungan terlihat jelas arahnya
import matplotlib as mpl
import matplotlib.pyplot as plt
import networkx as nx

seed = 13648
G = nx.random_k_out_graph (10,3,0.5, seed = seed)
pos = nx.spring_layout(G, seed=seed)

node_sizes = [3+10*i for i in range (len(G))]
M = G.number_of_edges()
```

```
edge_colors = range(2, M+2)
edge_alphas = [(5+i)/(M+4)for i in range (M)]
cmap = plt.cm.plasma

nodes = nx.draw_networkx_nodes(G,pos,node_size= node_sizes,node_color =␣
 ↪"indigo")
edges = nx.draw_networkx_edges(
    G,
    pos,
    node_size = node_sizes,
    arrowstyle= "->",
    arrowsize = 10,
    edge_color = edge_colors,
    edge_cmap = cmap,
    width = 2,
)

for i in range (M):
  edges[i].set_alpha(edge_alphas[i])

pc = mpl.collections.PatchCollection(edges, cmap=cmap)
pc.set_array(edge_colors)
plt.colorbar(pc, ax=plt.gca())

ax = plt.gca()

ax.set_axis_off()
plt.show()
```



[447]:
```
#Grafik ini menunjukkan rute dari traveller yang menunjukkan node node yang␣
 ↪saling terhubung
#Ketika hubungan node itu termasuk kedalam rute traveller itu (menggunakan␣
 ↪formula matematika yang dibuat), maka edge tersebut akan berwarna merah
#Node ditampilkan labelnya dan dengan bold
#Grafik ini berupa node node yang dibuat secara random dan radius edge tidak␣
 ↪lebih dari 0.4

import networkx.algorithms.approximation as nx_app
import math

G = nx.random_geometric_graph(20, radius=0.4, seed=3)
pos = nx.get_node_attributes(G, "pos")

pos[0] = (0.5, 0.5)
H = G.copy()

for i in range(len(pos)):
    for j in range(i+1, len(pos)):
        dist = math.hypot(pos[i][0] - pos[j][0], pos[i][1] - pos[j][1])
        dist = dist
        G.add_edge(i, j, weight=dist)
```

```
cycle = nx_app.christofides(G, weight="weight")
edge_list = list(nx.utils.pairwise(cycle))

nx.draw_networkx_edges(H, pos, edge_color="blue", width=0.5)

nx.draw_networkx(
    G,
    pos,
    with_labels=True,
    edgelist=edge_list,
    edge_color="red",
    node_size=200,
    width=3,
)

print("The route of the traveller is:", cycle)
plt.show()
```

```
The route of the traveller is: [0, 4, 19, 12, 2, 7, 10, 18, 5, 13, 6, 11, 3, 16,
17, 15, 14, 8, 9, 1, 0]
```