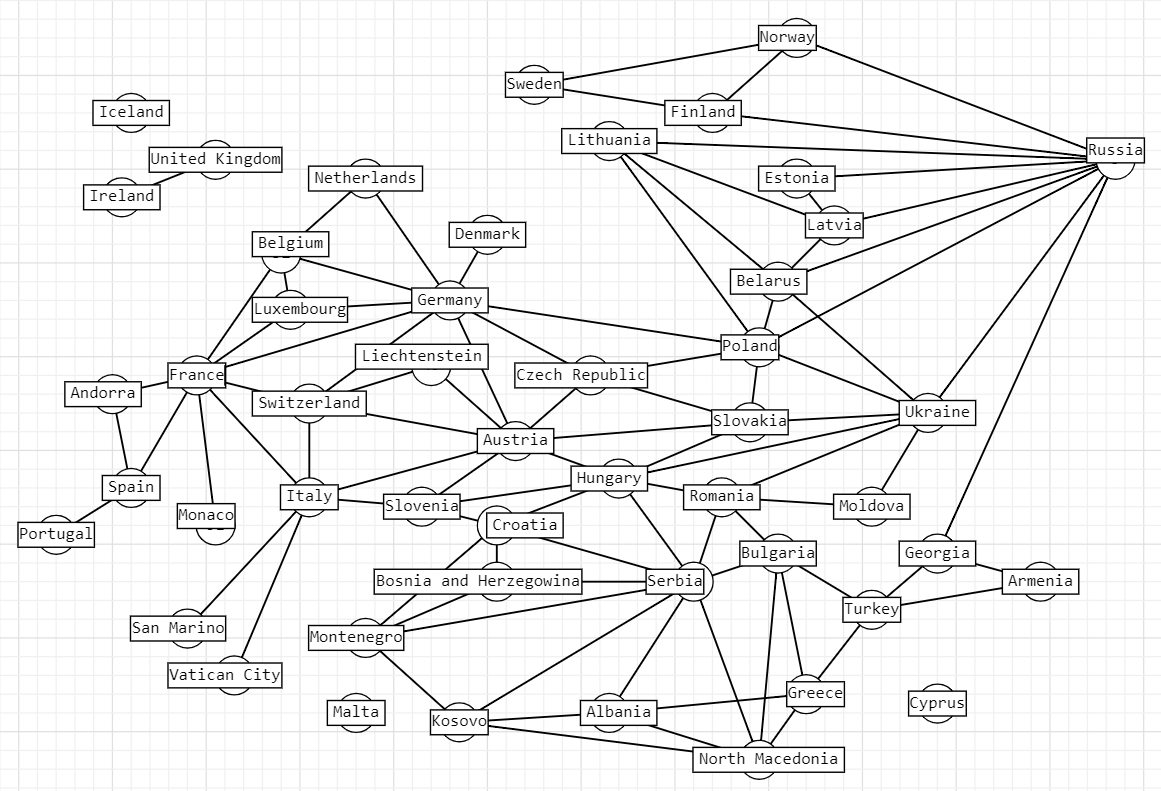
# HomeWork\_5 Петренко, М3104 Задание 1

Примечание: jupyter notebook для задания 1 и файл со списком ребер есть в репозитории по ссылке <https://github.com/petmila/My_graphs_discrete_math>

(a) Нарисовано с помощью сайта <https://programforyou.ru/graph-redactor>

добавлением вершин и ребер вручную.



# Список вершин и их географических координат:

geolock = {"Armenia":{"height": 40,"width": 44},

"Albania":{"height": 41,"width": 19},

"Andorra":{"height": 42,"width": 1},

"Austria":{"height": 48,"width": 16},

"Belarus":{"height": 53,"width": 27},

"Belgium":{"height": 50,"width": 4},

"Bosnia\_and\_Herzegovina":{"height": 43,"width": 18},

"Bulgaria":{"height": 42,"width": 23},

"Croatia":{"height": 45,"width": 15},

"Czech\_Republic":{"height": 50,"width": 14},

"Denmark":{"height": 55,"width": 12},

"Estonia":{"height": 59,"width": 24},

"Finland":{"height": 60,"width": 24},

"France":{"height": 48,"width": 2},

"Germany":{"height": 52,"width": 13},

"Georgia":{"height": 41,"width": 44},

"Greece":{"height": 37,"width": 23},

"Hungary":{"height": 47,"width": 19},

"Iceland":{"height": 64,"width": 21},

"Ireland":{"height": 53,"width": 6},

"Italy":{"height": 41,"width": 12},

"Kosovo":{"height": 42,"width": 21},

"Latvia":{"height": 56,"width": 24},

"Liechtenstein":{"height": 47,"width": 9},

"Lithuania":{"height": 54,"width": 25},

"Luxembourg":{"height": 49,"width": 6},

"Malta":{"height": 35,"width": 14},

"Moldova":{"height": 47,"width": 28},

"Monaco":{"height": 47,"width": 9},

"Montenegro":{"height": 42,"width": 19},

"Netherlands":{"height": 52,"width": 4},

"North\_Macedonia":{"height": 41,"width": 21},

"Norway":{"height": 59,"width": 10},

"Poland":{"height": 52,"width": 21},

"Portugal":{"height": 38,"width": 9},

"Romania":{"height": 47,"width": 19},

"Russia":{"height": 55,"width": 37},

"San\_Marino":{"height": 43,"width": 12},

"Serbia":{"height": 44,"width": 20},

"Slovakia":{"height": 48,"width": 17},

"Slovenia":{"height": 46,"width": 14},

"Spain":{"height": 40,"width": 3},

"Sweden":{"height": 59,"width": 18},

"Switzerland":{"height": 46,"width": 7},

"Turkey":{"height": 39,"width": 32},

"Ukraine":{"height": 50,"width": 30},

"United\_Kingdom":{"height": 51,"width": 0},

"Vatican\_City":{"height": 41,"width": 12},

}

import itertools

import matplotlib.pyplot as plt import networkx as nx

import numpy.random as rnd import haversine

**class** Europe:

**def** init (self):

*# создание графа по списку ребер из файла(файл есть в репозитории)*

self.graph = nx.read\_edgelist("graph.txt") *# добавляю островные государства* self.graph.add\_node('Malta') self.graph.add\_node('Iceland') self.graph.add\_node('Cyprus')

*# мощности множеств вершин и ребер графа* self.nodes\_number = self.graph.number\_of\_nodes() self.edges\_number = self.graph.number\_of\_edges()

*# поиск наибольшей компоненты связности из списка всех компонент*

subgraphs = list(nx.connected\_components(self.graph)) self.connected\_nodes\_list = {}

**for** subgraph **in** subgraphs:

**if** len(subgraph) > len(self.connected\_nodes\_list): self.connected\_nodes\_list = subgraph

self.largest\_connected\_component = nx.subgraph(self.graph, self.connected\_nodes\_list)

*# массив степеней всех вершин, чтобы найти минимальную и максимальную*

self.degree = nx.degree(self.largest\_connected\_component) min\_degree = self.nodes\_number + 1

max\_degree = 0

**for** node **in** self.degree:

**if** node[1] < min\_degree: min\_degree = node[1]

**if** node[1] > max\_degree: max\_degree = node[1] self.min\_degree = min\_degree self.max\_degree = max\_degree

*# раскраска - множество пар (вершина : номер цвета)*

self.v\_coloring = nx.greedy\_color(self.largest\_connected\_component)

*# создание дуального графа и получение раскраски для ребер тем же способом*

self.dual = nx.line\_graph(self.largest\_connected\_component) self.e\_coloring = nx.greedy\_color(self.dual)

cliques = list(nx.find\_cliques(self.largest\_connected\_component))

self.max\_clique = cliques[0]

**for** clique **in** cliques:

**if** len(self.max\_clique) < len(clique): self.max\_clique = clique

self.max\_stable\_set = nx.maximal\_independent\_set(self.largest\_connected\_component)

*# для невзвешенного графа дефолтны1 вес = 1, так что она посчитает просто количество ребер*

self.max\_matching = nx.max\_weight\_matching(self.largest\_connected\_component)

self.v\_cover = nx.approximation.min\_weighted\_vertex\_cover(self.largest\_connected\_comp onent)

self.e\_cover = nx.min\_edge\_cover(self.largest\_connected\_component)

self.blocks = list(nx.biconnected\_components(self.graph))

**def** number\_of\_nodes(self):

print("Number of nodes:", self.nodes\_number)

**def** number\_of\_edges(self):

print("Number of edges:", self.edges\_number)

**def** degree(self):

print("Minimum degree:", self.min\_degree) print("Maximum degree:", self.max\_degree)

**def** diameter(self): print("Diameter:",

nx.diameter(self.largest\_connected\_component))

**def** center(self):

print("Center:", nx.center(self.largest\_connected\_component))

**def** radius(self):

print("Radius:", nx.radius(self.largest\_connected\_component))

**def** girth(self):

*# поиск длины минимального цикла*

subgraphs = list(nx.minimum\_cycle\_basis(self.largest\_connected\_component))

nodes\_list = self.nodes\_number \* [0]

**for** subgraph **in** subgraphs:

**if** len(subgraph) < len(nodes\_list): nodes\_list = subgraph

print("Girth:", len(nodes\_list))

**def** vertex\_coloring(self): print("Minimum vertex coloring:",

max(self.v\_coloring.values()))

**def** edge\_coloring(self):

print("Minimum edge coloring:", max(self.e\_coloring.values()))

**def** maximum\_clique(self):

print("Maximum clique:", self.max\_clique)

**def** maximum\_stable\_set(self):

print("Maximum stable set:", self.max\_stable\_set)

**def** maximum\_matching(self):

print("Maximum matching:", self.max\_matching)

**def** minimum\_vertex\_cover(self):

print("Minimum vertex cover:", self.v\_cover)

**def** minimum\_edge\_cover(self):

print("Minimum edge cover:", self.e\_cover)

**def** shortest\_closed\_walk\_vertexes(self):

*# получаем список вершин*

self.tsp = (nx.algorithms.approximation.traveling\_salesman\_problem(self.largest\_c onnected\_component))

print("Shortest closed walk, which visits every vetrex:\n")

i = 0

*# вывод ребер, инцидентных вершинам из списка и самих вершин*

**while** i < (len(self.tsp) - 4):

print(self.tsp[i], (self.tsp[i], self.tsp[i + 1]),

self.tsp[i + 1], (self.tsp[i + 1], self.tsp[i + 2]),

self.tsp[i + 2], (self.tsp[i + 2], self.tsp[i + 3]),

self.tsp[i + 3], (self.tsp[i + 3], self.tsp[i + 4]))

i += 4

print(self.tsp[len(self.tsp) - 1])

**def** shortest\_closed\_walk\_edges(self):

print("Shortest closed walk, which visits every edge:\n")

*# eulerize добавляет кратные ребра так, чтобы в не эйлеровом графе можно было построить ейлеров путь*

self.list\_euler = list(nx.eulerian\_circuit(nx.eulerize(self.largest\_connected\_component)

))

i = 0

*# вывод ребер из списка и вершин, им инцидентных*

print(self.list\_euler[0][0])

**while** i < (len(self.list\_euler) - 4): print(self.list\_euler[i], self.list\_euler[i][1], self.list\_euler[i + 1], self.list\_euler[i + 1][1],

self.list\_euler[i + 2], self.list\_euler[i + 2][1],

self.list\_euler[i + 3], self.list\_euler[i + 3][1])

i += 4

**def** all\_blocks(self): print("Biconnected components:") **for** item **in** self.blocks:

print(item)

**def** all\_2\_edge\_connected\_components(self): print("2-edge-connected components:")

**for** item **in** list(nx.biconnected\_component\_edges(self.graph)): print(item)

**def** make\_weighted\_graph(self):

self.weighted\_graph = nx.read\_edgelist("graph.txt") *# создание графа по списку ребер из файла(файл есть в репозитории)*

self.weighted\_graph.add\_node('Malta') self.weighted\_graph.add\_node('Iceland') *# добавляю островные*

*государства*

self.weighted\_graph.add\_node('Cyprus')

nx.set\_node\_attributes(self.weighted\_graph, geolock) edges\_weight = {}

*# расчет весов как расстояний между географическими координатами городов*

**for** item **in** nx.edges(self.weighted\_graph):

edges\_weight[item] = int(haversine.haversine((self.weighted\_graph.nodes[item[0]]["height"], self.weighted\_graph.nodes[item[0]]["width"]),

(self.weighted\_graph.nodes[item[1]] ["height"],self.weighted\_graph.nodes[item[1]]["width"])))

nx.set\_edge\_attributes(self.weighted\_graph , edges\_weight, 'weight')

self.largest\_weighted\_connected\_component = nx.subgraph(self.weighted\_graph, self.connected\_nodes\_list)

**def** spanning\_tree(self): self.minimum\_spanning\_tree =

nx.minimum\_spanning\_tree(self.largest\_weighted\_connected\_component) print("Minimum\_spanning\_tree:")

**return** self.minimum\_spanning\_tree

**def** prufer(self): print("Prufer code:",

nx.to\_prufer\_sequence(nx.convert\_node\_labels\_to\_integers(self.minimum\_ spanning\_tree)))

# (b)

europe = Europe() europe.number\_of\_edges() europe.number\_of\_nodes() europe.degree() europe.diameter() europe.center() europe.radius() europe.girth()

Number of edges: 92 Number of nodes: 49 Diameter: 8

Center: ['Poland'] Radius: 4

Girth: 3

# Очевидно, что если в графе есть вершины, связанные с остальным графом только одним ребром, то:

(G) = 1 - минимальное число вершин, удалив которые, мы сделаем граф несвязным.

# (G) = 1 - минимальное число ребер, удалив которые, мы сделаем граф несвязным.

(c)

europe.vertex\_coloring() Minimum vertex coloring: 4 (d)

europe.edge\_coloring()

Minimum edge coloring: 8 (e) europe.maximum\_clique()

Maximum clique: ['Belarus', 'Russia', 'Lithuania', 'Latvia']

# (f)

europe.maximum\_stable\_set()

Maximum stable set: ['Russia', 'Czech\_Republic', 'Luxembourg', 'Montenegro', 'Slovenia', 'Romania', 'Armenia', 'Portugal', 'Denmark', 'North\_Macedonia', 'Netherlands', 'Andorra', 'Sweden', 'San\_Marino', 'Liechtenstein', 'Vatican\_City', 'Monaco']

# (g)

europe.maximum\_matching()

Maximum matching: {('Moldova', 'Romania'), ('Ukraine', 'Slovakia'), ('North\_Macedonia', 'Albania'), ('Lithuania', 'Belarus'), ('Montenegro', 'Bosnia\_and\_Herzegovina'), ('Croatia', 'Hungary'), ('Bulgaria', 'Greece'), ('Poland', 'Czech\_Republic'), ('Austria',

'Slovenia'), ('Estonia', 'Latvia'), ('Norway', 'Sweden'),

('Liechtenstein', 'Switzerland'), ('Turkey', 'Armenia'), ('Kosovo',

'Serbia'), ('Finland', 'Russia'), ('Monaco', 'France'),

('Vatican\_City', 'Italy'), ('Portugal', 'Spain'), ('Netherlands', 'Belgium'), ('Denmark', 'Germany')}

# (h)

europe.minimum\_vertex\_cover()

Minimum vertex cover: {'Belarus', 'Czech\_Republic', 'Croatia', 'Hungary', 'Greece', 'Slovenia', 'Spain', 'North\_Macedonia', 'Bulgaria', 'Austria', 'Ukraine', 'Serbia', 'Latvia', 'Germany', 'Romania', 'France', 'Switzerland', 'Finland', 'Georgia', 'Poland', 'Kosovo', 'Armenia', 'Belgium', 'Albania', 'Turkey', 'Slovakia', 'Bosnia\_and\_Herzegovina', 'Italy', 'Russia', 'Andorra', 'Sweden'}

# (i)

europe.minimum\_edge\_cover()

Minimum edge cover: {('Moldova', 'Romania'), ('Belgium', 'Luxembourg'), ('Ukraine', 'Slovakia'), ('North\_Macedonia', 'Albania'), ('Lithuania', 'Belarus'), ('Georgia', 'Armenia'), ('Andorra', 'Spain'), ('Montenegro', 'Bosnia\_and\_Herzegovina'), ('Luxembourg', 'Belgium'), ('Croatia', 'Hungary'), ('Armenia',

'Georgia'), ('Bulgaria', 'Greece'), ('Poland', 'Czech\_Republic'),

('Spain', 'Andorra'), ('Austria', 'Slovenia'), ('Estonia', 'Latvia'),

('Norway', 'Sweden'), ('Liechtenstein', 'Switzerland'), ('Italy',

'San\_Marino'), ('Turkey', 'Armenia'), ('Kosovo', 'Serbia'),

('San\_Marino', 'Italy'), ('Finland', 'Russia'), ('Monaco', 'France'),

('Vatican\_City', 'Italy'), ('Portugal', 'Spain'), ('Netherlands', 'Belgium'), ('Denmark', 'Germany')}

# (j)

europe.shortest\_closed\_walk\_vertexes()

Shortest closed walk, which visits every vetrex:

Armenia ('Armenia', 'Georgia') Georgia ('Georgia', 'Russia') Russia ('Russia', 'Norway') Norway ('Norway', 'Sweden')

Sweden ('Sweden', 'Finland') Finland ('Finland', 'Russia') Russia ('Russia', 'Estonia') Estonia ('Estonia', 'Latvia')

Latvia ('Latvia', 'Belarus') Belarus ('Belarus', 'Lithuania') Lithuania ('Lithuania', 'Poland') Poland ('Poland', 'Germany') Germany ('Germany', 'Denmark') Denmark ('Denmark', 'Germany') Germany ('Germany', 'Netherlands') Netherlands ('Netherlands', 'Belgium') Belgium ('Belgium', 'France') France ('France', 'Monaco') Monaco ('Monaco', 'France') France ('France', 'Luxembourg')

Luxembourg ('Luxembourg', 'Germany') Germany ('Germany', 'Czech\_Republic') Czech\_Republic ('Czech\_Republic', 'Austria') Austria ('Austria', 'Slovenia')

Slovenia ('Slovenia', 'Croatia') Croatia ('Croatia', 'Serbia') Serbia ('Serbia', 'Montenegro') Montenegro ('Montenegro', 'Bosnia\_and\_Herzegovina')

Bosnia\_and\_Herzegovina ('Bosnia\_and\_Herzegovina', 'Serbia') Serbia ('Serbia', 'Romania') Romania ('Romania', 'Moldova') Moldova ('Moldova', 'Ukraine')

Ukraine ('Ukraine', 'Slovakia') Slovakia ('Slovakia', 'Austria') Austria ('Austria', 'Liechtenstein') Liechtenstein ('Liechtenstein', 'Switzerland')

Switzerland ('Switzerland', 'Italy') Italy ('Italy', 'Vatican\_City') Vatican\_City ('Vatican\_City', 'Italy') Italy ('Italy', 'San\_Marino') San\_Marino ('San\_Marino', 'Italy') Italy ('Italy', 'France') France ('France', 'Spain') Spain ('Spain', 'Portugal')

Portugal ('Portugal', 'Spain') Spain ('Spain', 'Andorra') Andorra ('Andorra', 'France') France ('France', 'Germany')

Germany ('Germany', 'Austria') Austria ('Austria', 'Hungary') Hungary ('Hungary', 'Serbia') Serbia ('Serbia', 'Albania')

Albania ('Albania', 'North\_Macedonia') North\_Macedonia ('North\_Macedonia', 'Kosovo') Kosovo ('Kosovo', 'Albania') Albania

('Albania', 'Greece')

Greece ('Greece', 'Turkey') Turkey ('Turkey', 'Bulgaria') Bulgaria ('Bulgaria', 'Turkey') Turkey ('Turkey', 'Armenia')

Armenia

# (k)

europe.shortest\_closed\_walk\_edges()

Shortest closed walk, which visits every edge:

Armenia

('Armenia', 'Georgia') Georgia ('Georgia', 'Russia') Russia ('Russia', 'Ukraine') Ukraine ('Ukraine', 'Romania') Romania

('Romania', 'Moldova') Moldova ('Moldova', 'Ukraine') Ukraine ('Ukraine', 'Hungary') Hungary ('Hungary', 'Slovenia') Slovenia ('Slovenia', 'Italy') Italy ('Italy', 'Vatican\_City') Vatican\_City ('Vatican\_City', 'Italy') Italy ('Italy', 'San\_Marino') San\_Marino ('San\_Marino', 'Italy') Italy ('Italy', 'Switzerland') Switzerland ('Switzerland', 'Liechtenstein') Liechtenstein ('Liechtenstein', 'Austria') Austria

('Austria', 'Switzerland') Switzerland ('Switzerland', 'Austria') Austria ('Austria', 'Slovenia') Slovenia ('Slovenia', 'Croatia')

Croatia

('Croatia', 'Montenegro') Montenegro ('Montenegro', 'Bosnia\_and\_Herzegovina') Bosnia\_and\_Herzegovina ('Bosnia\_and\_Herzegovina', 'Croatia') Croatia ('Croatia', 'Hungary')

Hungary

('Hungary', 'Ukraine') Ukraine ('Ukraine', 'Poland') Poland ('Poland', 'Lithuania') Lithuania ('Lithuania', 'Latvia') Latvia

('Latvia', 'Russia') Russia ('Russia', 'Norway') Norway ('Norway', 'Finland') Finland ('Finland', 'Norway') Norway

('Norway', 'Sweden') Sweden ('Sweden', 'Finland') Finland ('Finland', 'Russia') Russia ('Russia', 'Lithuania') Lithuania

('Lithuania', 'Belarus') Belarus ('Belarus', 'Latvia') Latvia ('Latvia', 'Estonia') Estonia ('Estonia', 'Russia') Russia

('Russia', 'Poland') Poland ('Poland', 'Belarus') Belarus ('Belarus', 'Poland') Poland ('Poland', 'Slovakia') Slovakia

('Slovakia', 'Hungary') Hungary ('Hungary', 'Romania') Romania ('Romania', 'Serbia') Serbia ('Serbia', 'Romania') Romania ('Romania', 'Bulgaria') Bulgaria ('Bulgaria', 'North\_Macedonia') North\_Macedonia ('North\_Macedonia', 'Kosovo') Kosovo ('Kosovo', 'Montenegro') Montenegro

('Montenegro', 'Serbia') Serbia ('Serbia', 'North\_Macedonia') North\_Macedonia ('North\_Macedonia', 'Bulgaria') Bulgaria ('Bulgaria', 'Serbia') Serbia

('Serbia', 'Croatia') Croatia ('Croatia', 'Bosnia\_and\_Herzegovina') Bosnia\_and\_Herzegovina ('Bosnia\_and\_Herzegovina', 'Serbia') Serbia ('Serbia', 'Kosovo') Kosovo

('Kosovo', 'Albania') Albania ('Albania', 'North\_Macedonia') North\_Macedonia ('North\_Macedonia', 'Greece') Greece ('Greece',

'Albania') Albania

('Albania', 'Serbia') Serbia ('Serbia', 'Hungary') Hungary ('Hungary', 'Austria') Austria ('Austria', 'Italy') Italy

('Italy', 'France') France ('France', 'Monaco') Monaco ('Monaco', 'France') France ('France', 'Luxembourg') Luxembourg ('Luxembourg', 'Belgium') Belgium ('Belgium', 'Netherlands') Netherlands ('Netherlands', 'Germany') Germany ('Germany', 'Luxembourg') Luxembourg

('Luxembourg', 'France') France ('France', 'Switzerland') Switzerland ('Switzerland', 'Germany') Germany ('Germany', 'Denmark') Denmark ('Denmark', 'Germany') Germany ('Germany', 'Poland') Poland ('Poland', 'Czech\_Republic') Czech\_Republic ('Czech\_Republic', 'Slovakia')

Slovakia

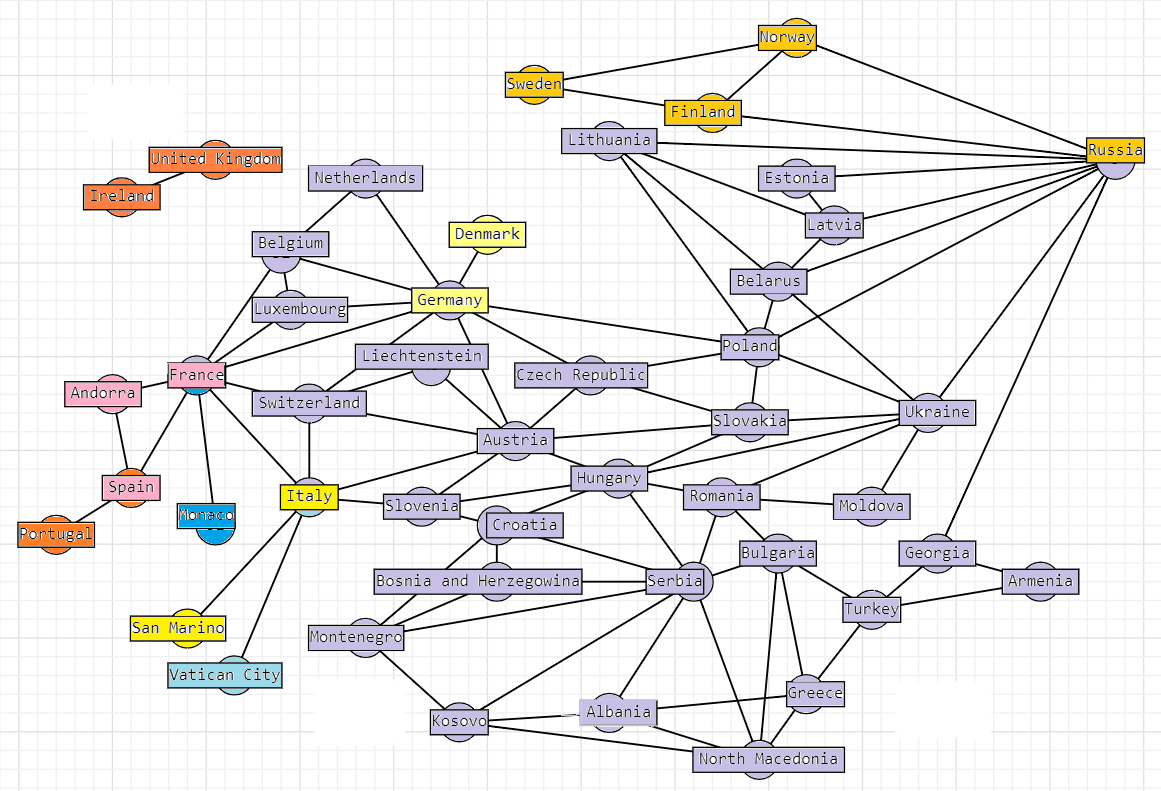
('Slovakia', 'Austria') Austria ('Austria', 'Czech\_Republic') Czech\_Republic ('Czech\_Republic', 'Germany') Germany ('Germany', 'Belgium') Belgium

('Belgium', 'France') France ('France', 'Spain') Spain ('Spain', 'Portugal') Portugal ('Portugal', 'Spain') Spain

('Spain', 'Andorra') Andorra ('Andorra', 'France') France ('France', 'Germany') Germany ('Germany', 'Austria') Austria

('Austria', 'Slovakia') Slovakia ('Slovakia', 'Ukraine') Ukraine ('Ukraine', 'Belarus') Belarus ('Belarus', 'Russia') Russia ('Russia', 'Georgia') Georgia ('Georgia', 'Turkey') Turkey ('Turkey', 'Greece') Greece ('Greece', 'Bulgaria') Bulgaria

# (l)



europe.all\_blocks() Biconnected components:

{'Spain', 'Portugal'}

{'Spain', 'France', 'Andorra'}

{'Italy', 'San\_Marino'}

{'Vatican\_City', 'Italy'}

{'France', 'Monaco'}

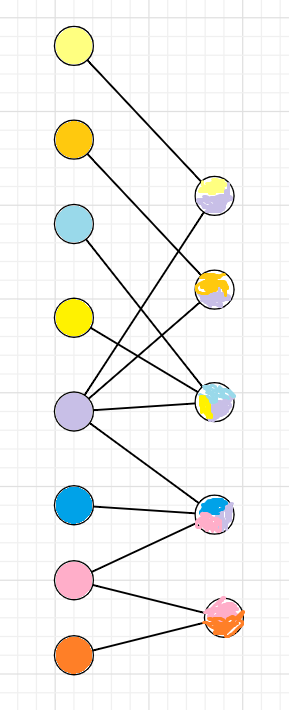
{'Norway', 'Russia', 'Finland', 'Sweden'}

{'Germany', 'Denmark'}

{'Belarus', 'Czech\_Republic', 'Luxembourg', 'Croatia', 'Hungary', 'Lithuania', 'Greece', 'Slovenia', 'Netherlands', 'North\_Macedonia', 'Bulgaria', 'Austria', 'Ukraine', 'Serbia', 'Latvia', 'Liechtenstein', 'Estonia', 'Germany', 'Montenegro', 'Romania', 'France', 'Switzerland', 'Poland', 'Kosovo', 'Belgium', 'Armenia', 'Albania', 'Turkey', 'Slovakia', 'Bosnia\_and\_Herzegovina', 'Italy', 'Russia', 'Moldova', 'Georgia'}

{'Ireland', 'United\_Kingdom'}

# Block-cut tree:



(m)

europe.all\_2\_edge\_connected\_components() 2-edge-connected components:

[('Spain', 'Portugal')]

[('France', 'Andorra'), ('Andorra', 'Spain'), ('Spain', 'France')] [('Italy', 'San\_Marino')]

[('Italy', 'Vatican\_City')]

[('France', 'Monaco')]

[('Russia', 'Finland'), ('Finland', 'Sweden'), ('Sweden', 'Norway'), ('Norway', 'Finland'), ('Norway', 'Russia')]

[('Germany', 'Denmark')]

[('Armenia', 'Turkey'), ('Turkey', 'Bulgaria'), ('Bulgaria',

'Greece'), ('Greece', 'Albania'), ('Albania', 'Serbia'), ('Serbia', 'Bosnia\_and\_Herzegovina'), ('Bosnia\_and\_Herzegovina', 'Croatia'), ('Croatia', 'Hungary'), ('Hungary', 'Austria'), ('Austria',

'Germany'), ('Germany', 'Belgium'), ('Belgium', 'France'), ('France',

'Luxembourg'), ('Luxembourg', 'Belgium'), ('Luxembourg', 'Germany'),

('France', 'Germany'), ('France', 'Switzerland'), ('Switzerland',

'Austria'), ('Switzerland', 'Germany'), ('Switzerland', 'Italy'),

('Italy', 'Austria'), ('Italy', 'France'), ('Italy', 'Slovenia'),

('Slovenia', 'Austria'), ('Slovenia', 'Croatia'), ('Slovenia', 'Hungary'), ('Switzerland', 'Liechtenstein'), ('Liechtenstein', 'Austria'), ('Belgium', 'Netherlands'), ('Netherlands', 'Germany'), ('Germany', 'Czech\_Republic'), ('Czech\_Republic', 'Austria'), ('Czech\_Republic', 'Slovakia'), ('Slovakia', 'Austria'), ('Slovakia',

'Hungary'), ('Slovakia', 'Poland'), ('Poland', 'Belarus'), ('Belarus',

'Russia'), ('Russia', 'Estonia'), ('Estonia', 'Latvia'), ('Latvia',

'Belarus'), ('Latvia', 'Lithuania'), ('Lithuania', 'Belarus'),

('Lithuania', 'Poland'), ('Lithuania', 'Russia'), ('Latvia',

'Russia'), ('Russia', 'Georgia'), ('Georgia', 'Armenia'), ('Georgia',

'Turkey'), ('Russia', 'Poland'), ('Russia', 'Ukraine'), ('Ukraine',

'Belarus'), ('Ukraine', 'Hungary'), ('Ukraine', 'Moldova'),

('Moldova', 'Romania'), ('Romania', 'Bulgaria'), ('Romania',

'Hungary'), ('Romania', 'Serbia'), ('Romania', 'Ukraine'), ('Ukraine',

'Poland'), ('Ukraine', 'Slovakia'), ('Poland', 'Czech\_Republic'),

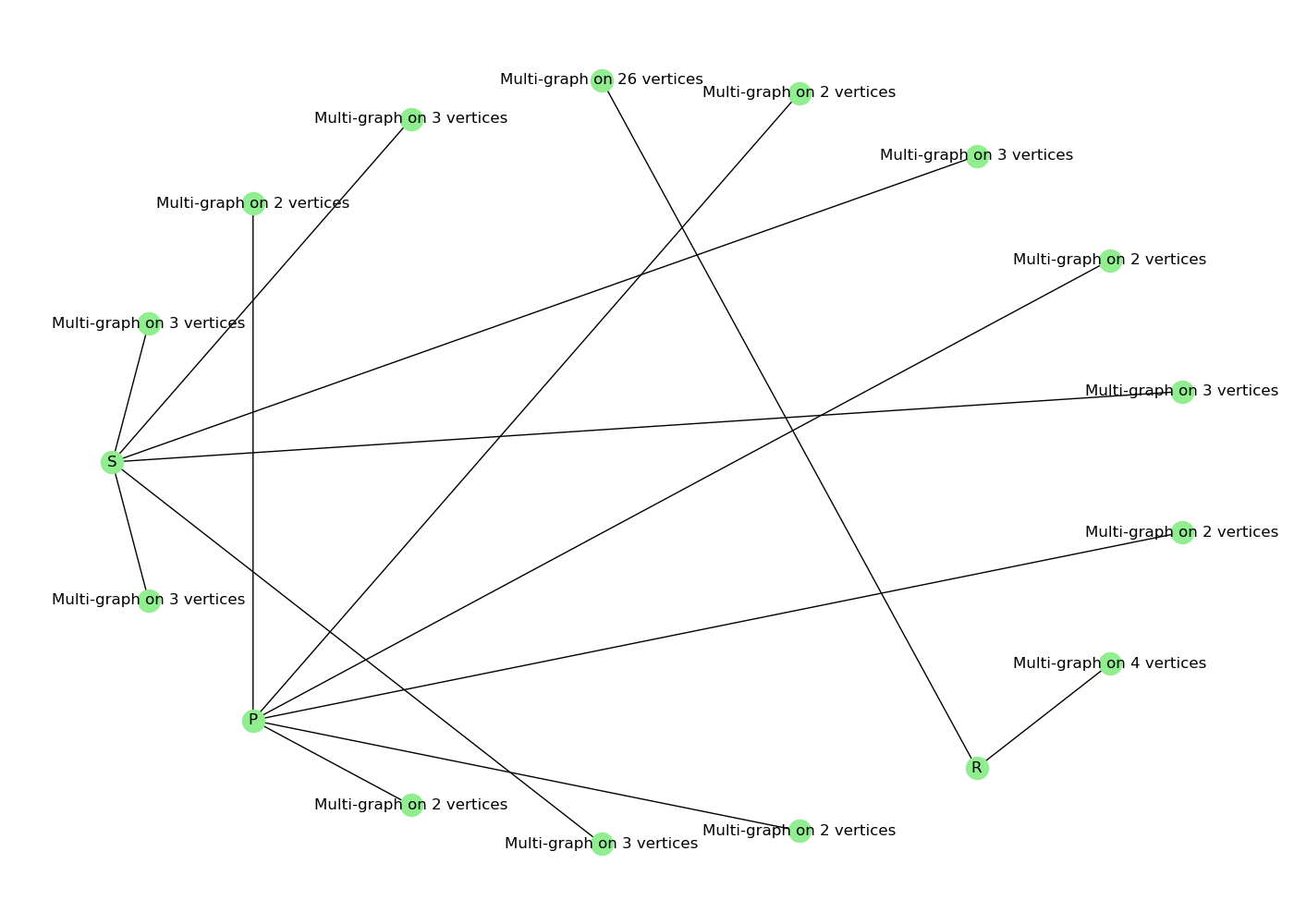
('Poland', 'Germany'), ('Hungary', 'Serbia'), ('Croatia', 'Serbia'), ('Croatia', 'Montenegro'), ('Montenegro', 'Bosnia\_and\_Herzegovina'), ('Montenegro', 'Kosovo'), ('Kosovo', 'Albania'), ('Kosovo', 'Serbia'), ('Kosovo', 'North\_Macedonia'), ('North\_Macedonia', 'Albania'), ('North\_Macedonia', 'Bulgaria'), ('North\_Macedonia', 'Greece'), ('North\_Macedonia', 'Serbia'), ('Montenegro', 'Serbia'), ('Serbia', 'Bulgaria'), ('Greece', 'Turkey')]

[('Ireland', 'United\_Kingdom')]

# (n)

SPQR - дерево. Было нарисовано в sagemathcell тут: [https://sagecell.sagemath.org/?](https://sagecell.sagemath.org/?z=eJx9VE1r3DAQvS_sfxC52AGzNNBTIIc0tJvSpJQ0PQVjtPbEFitrXH3sV-l_70jxh3a37cn2zLw3b97IetXYMsNrWNSad41ZlKgUlFZshN0z0XaoLTPdT11YDTCfWXbDlr4y_ZXc6haU4Mk1e0menV7DPsmSJaCuKZhnya1c8ZB_SZYELoHS30GvKJQlX9DgBunlK3VoikdeQoWqBzpjNb0yz7wE3XLlqe8OUDbFE3RuJUXpySRu-DrQ3TtVc73vg0FWlny2XIbQVtgDaMlVRV8PgmgsKGNBKGo3nyUfQHLtjFf65IwJ4B9rzYXymr_hgOR2E3IPwjaO92oJXAvXevAnzVUYcxL94HbQrtDp2s8KtnnTYQISDXEU9Fncgz5AjRuh3vy808ht6DU69oiKVEOtMWCdpIGDSbH7g8-nrmbJE7ZcxYxh8qEPkUQW_qVlZGx-torreE3RVnrnCPDR2KCDxRb3flK6N46SR35FpPEKSffbavOphKDjnk72_Z8NDKc1VtWbmY9uUvLczqlq8I3KotGn43Nm_JGV4ywefqQ79rv_W3zNwHEu6fiA9OZ6w6fjmg1j5qf_ATvpn8enPDZ3InhEWdG8wbxxxmHuMFmkaJKeZ-xMPIvTQ69_GjoqGNqy2JhJwVBJ2Sg4csbh35fz2TOtm6638bZLLQX7K1CB3aJe7xg3TO3GcMttJ9FKsVp0e__m852081lFTGq3sFgM0CLcsKnvQrxUtHgVtdOQ0sOIA9ykV-8zdvXukrKErDTfFqYBKdMqo8U0heQrkObmWTvImMIKihIlamp0IUXd2JqY1UXPbRrcppd_AGj9-e0%3D&lang=sage&interacts=eJyLjgUAARUAuQ%3D%3D) [z=eJx9VE1r3DAQvS\_sfxC52AGzNNBTIIc0tJvSpJQ0PQVjtPbEFitrXH3sV- l\_70jxh3a37cn2zLw3b97IetXYMsNrWNSad41ZlKgUlFZshN0z0XaoLTPdT11YDTCfWXbDlr4](https://sagecell.sagemath.org/?z=eJx9VE1r3DAQvS_sfxC52AGzNNBTIIc0tJvSpJQ0PQVjtPbEFitrXH3sV-l_70jxh3a37cn2zLw3b97IetXYMsNrWNSad41ZlKgUlFZshN0z0XaoLTPdT11YDTCfWXbDlr4y_ZXc6haU4Mk1e0menV7DPsmSJaCuKZhnya1c8ZB_SZYELoHS30GvKJQlX9DgBunlK3VoikdeQoWqBzpjNb0yz7wE3XLlqe8OUDbFE3RuJUXpySRu-DrQ3TtVc73vg0FWlny2XIbQVtgDaMlVRV8PgmgsKGNBKGo3nyUfQHLtjFf65IwJ4B9rzYXymr_hgOR2E3IPwjaO92oJXAvXevAnzVUYcxL94HbQrtDp2s8KtnnTYQISDXEU9Fncgz5AjRuh3vy808ht6DU69oiKVEOtMWCdpIGDSbH7g8-nrmbJE7ZcxYxh8qEPkUQW_qVlZGx-torreE3RVnrnCPDR2KCDxRb3flK6N46SR35FpPEKSffbavOphKDjnk72_Z8NDKc1VtWbmY9uUvLczqlq8I3KotGn43Nm_JGV4ywefqQ79rv_W3zNwHEu6fiA9OZ6w6fjmg1j5qf_ATvpn8enPDZ3InhEWdG8wbxxxmHuMFmkaJKeZ-xMPIvTQ69_GjoqGNqy2JhJwVBJ2Sg4csbh35fz2TOtm6638bZLLQX7K1CB3aJe7xg3TO3GcMttJ9FKsVp0e__m852081lFTGq3sFgM0CLcsKnvQrxUtHgVtdOQ0sOIA9ykV-8zdvXukrKErDTfFqYBKdMqo8U0heQrkObmWTvImMIKihIlamp0IUXd2JqY1UXPbRrcppd_AGj9-e0%3D&lang=sage&interacts=eJyLjgUAARUAuQ%3D%3D)

# [y\_ZXc6haU4Mk1e0menV7DPsmSJaCuKZhnya1c8ZB\_SZYELoHS30GvKJQlX9DgBunlK3Voikd eQoWqBzpjNb0yz7wE3XLlqe8OUDbFE3RuJUXpySRu- DrQ3TtVc73vg0FWlny2XIbQVtgDaMlVRV8PgmgsKGNBKGo3nyUfQHLtjFf65IwJ4B9rzYXy mr\_hgOR2E3IPwjaO92oJXAvXevAnzVUYcxL94HbQrtDp2s8KtnnTYQISDXEU9Fncgz5AjRuh 3vy808ht6DU69oiKVEOtMWCdpIGDSbH7g8-nrmbJE7ZcxYxh8qEPkUQW\_qVlZGx- torreE3RVnrnCPDR2KCDxRb3flK6N46SR35FpPEKSffbavOphKDjnk72\_Z8NDKc1VtWbmY9 uUvLczqlq8I3KotGn43Nm\_JGV4ywefqQ79rv\_W3zNwHEu6fiA9OZ6w6fjmg1j5qf\_ATvpn8en PDZ3InhEWdG8wbxxxmHuMFmkaJKeZ- xMPIvTQ69\_GjoqGNqy2JhJwVBJ2Sg4csbh35fz2TOtm6638bZLLQX7K1CB3aJe7xg3TO3GcM ttJ9FKsVp0e m852081lFTGq3sFgM0CLcsKnvQrxUtHgVtdOQ0sOIA9ykV- 8zdvXukrKErDTfFqYBKdMqo8U0heQrkObmWTvImMIKihIlamp0IUXd2JqY1UXPbRrcppd\_A](https://sagecell.sagemath.org/?z=eJx9VE1r3DAQvS_sfxC52AGzNNBTIIc0tJvSpJQ0PQVjtPbEFitrXH3sV-l_70jxh3a37cn2zLw3b97IetXYMsNrWNSad41ZlKgUlFZshN0z0XaoLTPdT11YDTCfWXbDlr4y_ZXc6haU4Mk1e0menV7DPsmSJaCuKZhnya1c8ZB_SZYELoHS30GvKJQlX9DgBunlK3VoikdeQoWqBzpjNb0yz7wE3XLlqe8OUDbFE3RuJUXpySRu-DrQ3TtVc73vg0FWlny2XIbQVtgDaMlVRV8PgmgsKGNBKGo3nyUfQHLtjFf65IwJ4B9rzYXymr_hgOR2E3IPwjaO92oJXAvXevAnzVUYcxL94HbQrtDp2s8KtnnTYQISDXEU9Fncgz5AjRuh3vy808ht6DU69oiKVEOtMWCdpIGDSbH7g8-nrmbJE7ZcxYxh8qEPkUQW_qVlZGx-torreE3RVnrnCPDR2KCDxRb3flK6N46SR35FpPEKSffbavOphKDjnk72_Z8NDKc1VtWbmY9uUvLczqlq8I3KotGn43Nm_JGV4ywefqQ79rv_W3zNwHEu6fiA9OZ6w6fjmg1j5qf_ATvpn8enPDZ3InhEWdG8wbxxxmHuMFmkaJKeZ-xMPIvTQ69_GjoqGNqy2JhJwVBJ2Sg4csbh35fz2TOtm6638bZLLQX7K1CB3aJe7xg3TO3GcMttJ9FKsVp0e__m852081lFTGq3sFgM0CLcsKnvQrxUtHgVtdOQ0sOIA9ykV-8zdvXukrKErDTfFqYBKdMqo8U0heQrkObmWTvImMIKihIlamp0IUXd2JqY1UXPbRrcppd_AGj9-e0%3D&lang=sage&interacts=eJyLjgUAARUAuQ%3D%3D) [Gj9-e0=&lang=sage&interacts=eJyLjgUAARUAuQ==](https://sagecell.sagemath.org/?z=eJx9VE1r3DAQvS_sfxC52AGzNNBTIIc0tJvSpJQ0PQVjtPbEFitrXH3sV-l_70jxh3a37cn2zLw3b97IetXYMsNrWNSad41ZlKgUlFZshN0z0XaoLTPdT11YDTCfWXbDlr4y_ZXc6haU4Mk1e0menV7DPsmSJaCuKZhnya1c8ZB_SZYELoHS30GvKJQlX9DgBunlK3VoikdeQoWqBzpjNb0yz7wE3XLlqe8OUDbFE3RuJUXpySRu-DrQ3TtVc73vg0FWlny2XIbQVtgDaMlVRV8PgmgsKGNBKGo3nyUfQHLtjFf65IwJ4B9rzYXymr_hgOR2E3IPwjaO92oJXAvXevAnzVUYcxL94HbQrtDp2s8KtnnTYQISDXEU9Fncgz5AjRuh3vy808ht6DU69oiKVEOtMWCdpIGDSbH7g8-nrmbJE7ZcxYxh8qEPkUQW_qVlZGx-torreE3RVnrnCPDR2KCDxRb3flK6N46SR35FpPEKSffbavOphKDjnk72_Z8NDKc1VtWbmY9uUvLczqlq8I3KotGn43Nm_JGV4ywefqQ79rv_W3zNwHEu6fiA9OZ6w6fjmg1j5qf_ATvpn8enPDZ3InhEWdG8wbxxxmHuMFmkaJKeZ-xMPIvTQ69_GjoqGNqy2JhJwVBJ2Sg4csbh35fz2TOtm6638bZLLQX7K1CB3aJe7xg3TO3GcMttJ9FKsVp0e__m852081lFTGq3sFgM0CLcsKnvQrxUtHgVtdOQ0sOIA9ykV-8zdvXukrKErDTfFqYBKdMqo8U0heQrkObmWTvImMIKihIlamp0IUXd2JqY1UXPbRrcppd_AGj9-e0%3D&lang=sage&interacts=eJyLjgUAARUAuQ%3D%3D)



(o)

europe.make\_weighted\_graph() MST = europe.spanning\_tree()

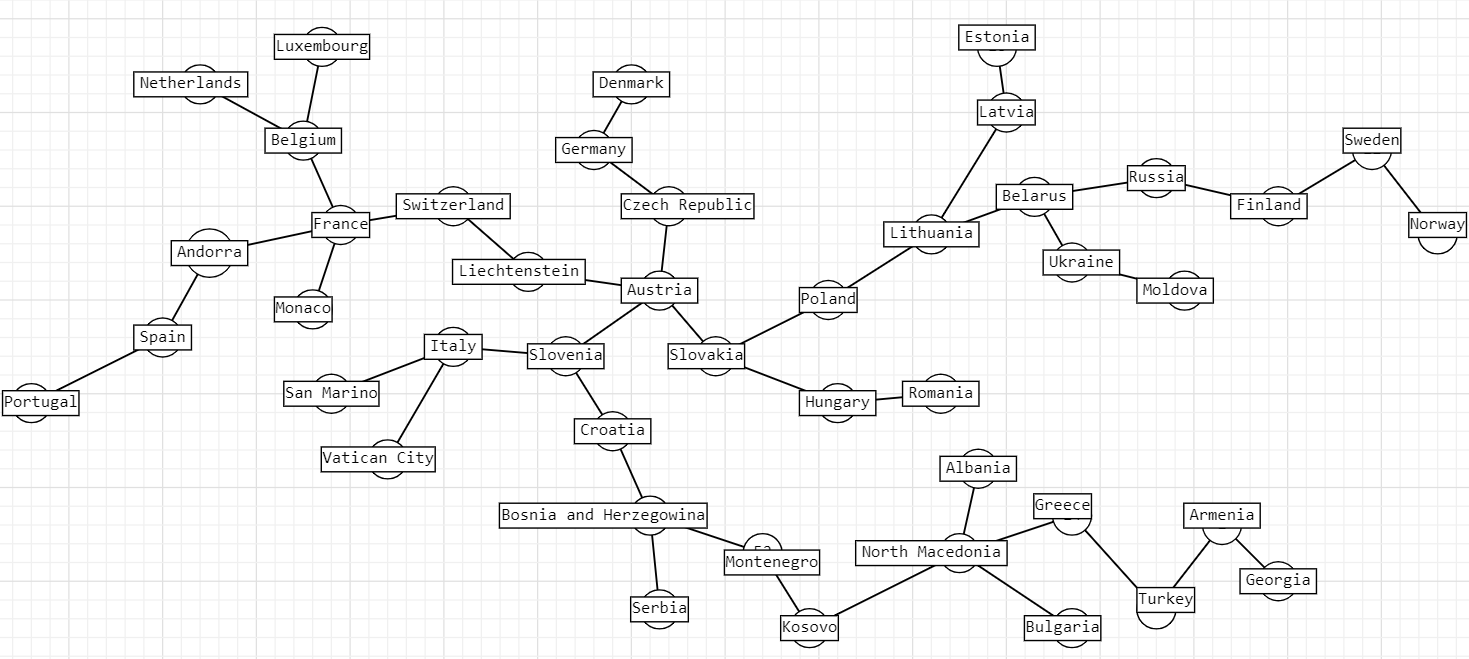
*#for i in MST.nodes(): # можно вывести список ребер для каждой вершины, но он очень большой, поэтому я не стала его здесь приводить # print(MST.edges(i))*

print(MST)

Minimum\_spanning\_tree:

Graph with 44 nodes and 43 edges

# (p) Минимальное остовное дерево было изображено вручную с помощью онлайн редактора графов по списку ребер для каждой вершины, полученному в предыдущем пункте.



Из рисунка видно, что центроидом является Austria. (q)

europe.prufer()

Prufer code: [0, 1, 4, 7, 7, 29, 26, 26, 10, 7, 6, 31, 29, 30, 16, 15,

14, 12, 13, 11, 24, 25, 37, 36, 21, 20, 10, 17, 17, 16, 11, 22, 20,

25, 23, 14, 11, 19, 18, 10, 8, 9]