

Homework #5

Graph Theory

Suslov Misha, M3104

<https://github.com/SuSFCTV/IS-ITMO-DM>

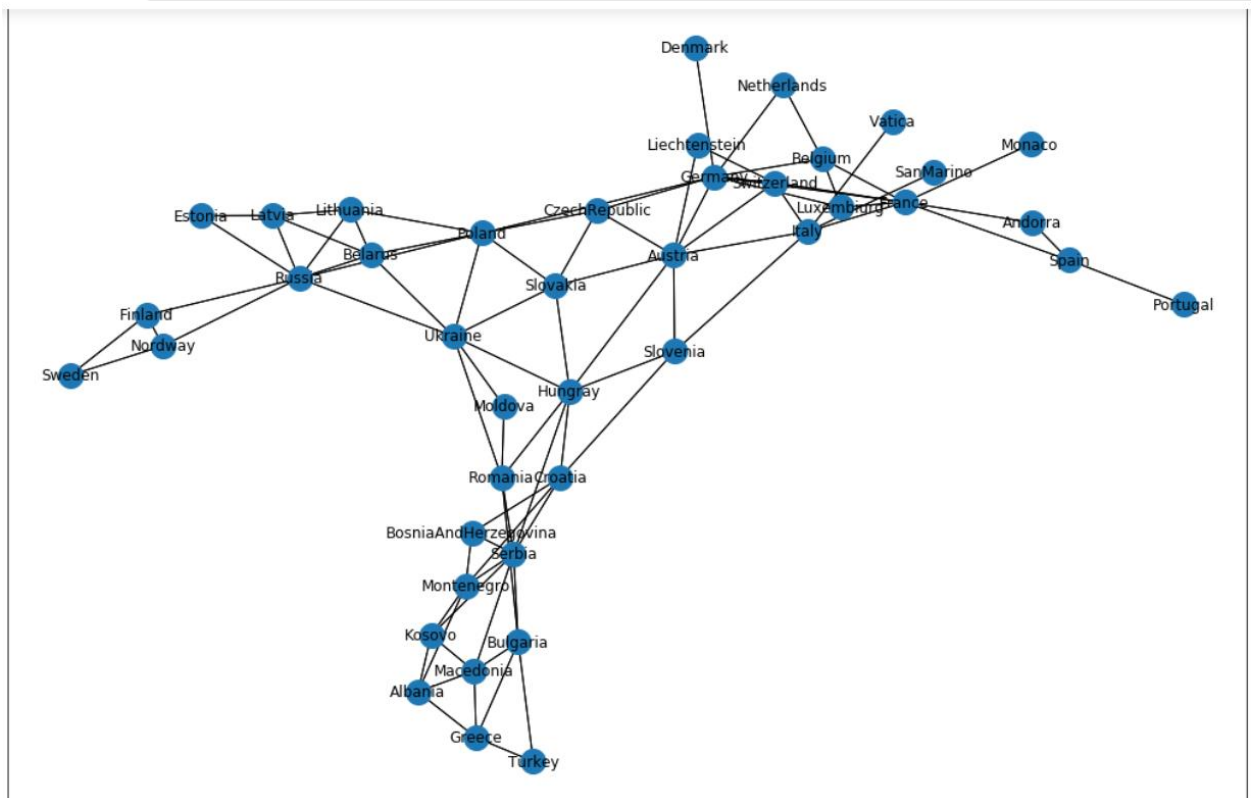
Задание №1.

A : Prove that G is planar by drawing it on a plane without an intersection of edges.

Плоский граф – граф, который можно изобразить на плоскости без пересечения ребёр не по вершинам. То есть, его вершины – точки на плоскости, а рёбра – непересекающиеся кривые на ней.

```
In [5]: # drawing graph
plt.figure(figsize=(18, 12))

pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'), node_size=400)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_edges(G, pos, arrows=False)
plt.show()
```



Видно, что изображённый граф удовлетворяет определению плоского графа, т.е. условиям, следовательно является плоским.

B : Find $|V|$, $|E|$, $\delta(G)$, $\Delta(G)$, $\text{rad}(G)$, $\text{diam}(G)$, $\text{girth}(G)$, $\text{center}(G)$, $\kappa(G)$, $\lambda(G)$.

$|V| = 42$ – число вершин графа

$|E| = 87$ – кол-во рёбер графа

Степень вершины:

```
degree = [0] * V
for i in range(V):
    degree[i] = sum(matrix[i])

print('max_degree =', max(degree))
print('min_degree =', min(degree))

max_degree = 9
min_degree = 1
```

$\delta(G) = 1$

$\Delta(G) = 9$

Радиус, диаметр, центр:

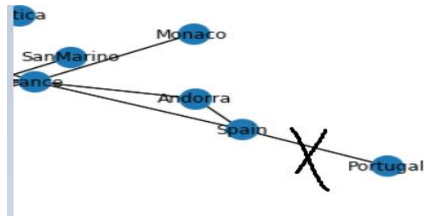
```
# Calculate the shortest distance using the Floyd-Warshall algorithm
distance = [[0] * V for i in range(V)]
inf = 10 ** 5
for i in range(V):
    for j in range(V):
        if matrix[i][j]:
            distance[i][j] = 1
        else:
            distance[i][j] = inf
for k in range(V):
    for i in range(V):
        for j in range(V):
            distance[i][j] = min(distance[i][j], distance[i][k] + distance[k][j])
```

```
ext = [0] * V
for i in range(V):
    for j in range(V):
        ext[i] = max(ext[i], distance[i][j])
```

```
# finding rad, diam ,cent
rad = min(ext)
diam = max(ext)
center = []
for i in range(V):
    if ext[i] == rad:
        center.append(country[i])
```

```
radius = 5
diameter = 8
center = ['Austria', 'Belarus', 'Croatia', 'CzechRepublic', 'Germany', 'Hungary', 'Poland', 'Russia', 'Slovakia', 'Slovenia', 'Switzerland', 'Ukraine']
```

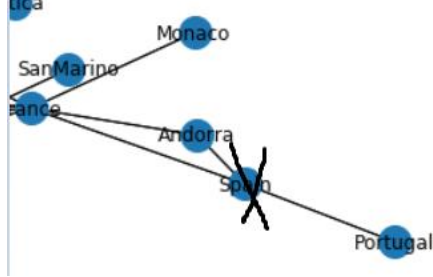
Рёберная связность $\lambda(G)$:



Убираем ребро между Испанией и Португалией –

получаем несвязный граф. $\lambda(G) = 1$

Вершинная связность $\kappa(G)$:



Убираем вершину с Испанией – получаем несвязный

граф. $\kappa(G) = 1$

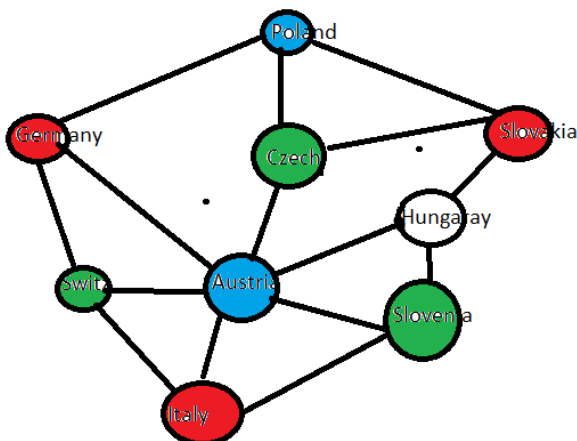
C: Find the minimum vertex coloring $Z : V \rightarrow \mathbb{N}$ of G .

Правильная вершинная раскраска – никакие две смежные вершины в ней не получают одного цвета.



Данный подграф – полный граф, сл-но минимальное кол-во цветов для построения вершинной раскраски больше или равно 3.

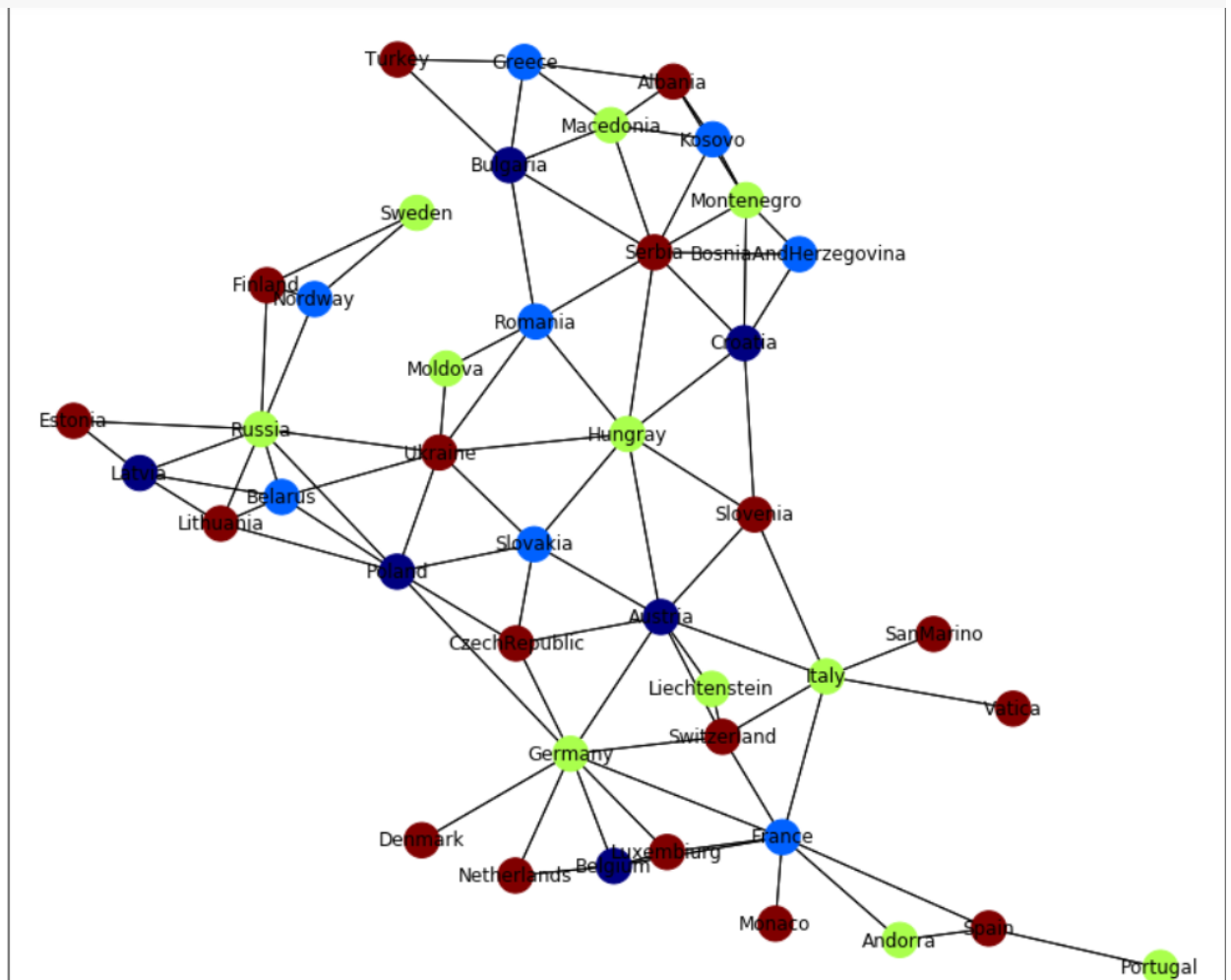
Докажем, что минимальное кол-во цветов будет равно 4:



Пытаемся раскрасить этот граф в три цвета. Рассмотрим вершину Hungary. Она смежна с вершинами, которые имеют три различных цвета : Austria –синий, Slovenia – зелёный и Slovakia – красный. Из этого сделаем вывод, что ей нельзя присвоить ни один из данных трёх цветов. При попытках раскрасить в другие три цвета, результат будет такой же.

С помощью bfs раскрасим граф 4-мя цветами:

```
plt.figure(figsize=(14, 12))
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'), node_size=400, node_color=val)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_edges(G, pos, arrows=False)
plt.show()
```

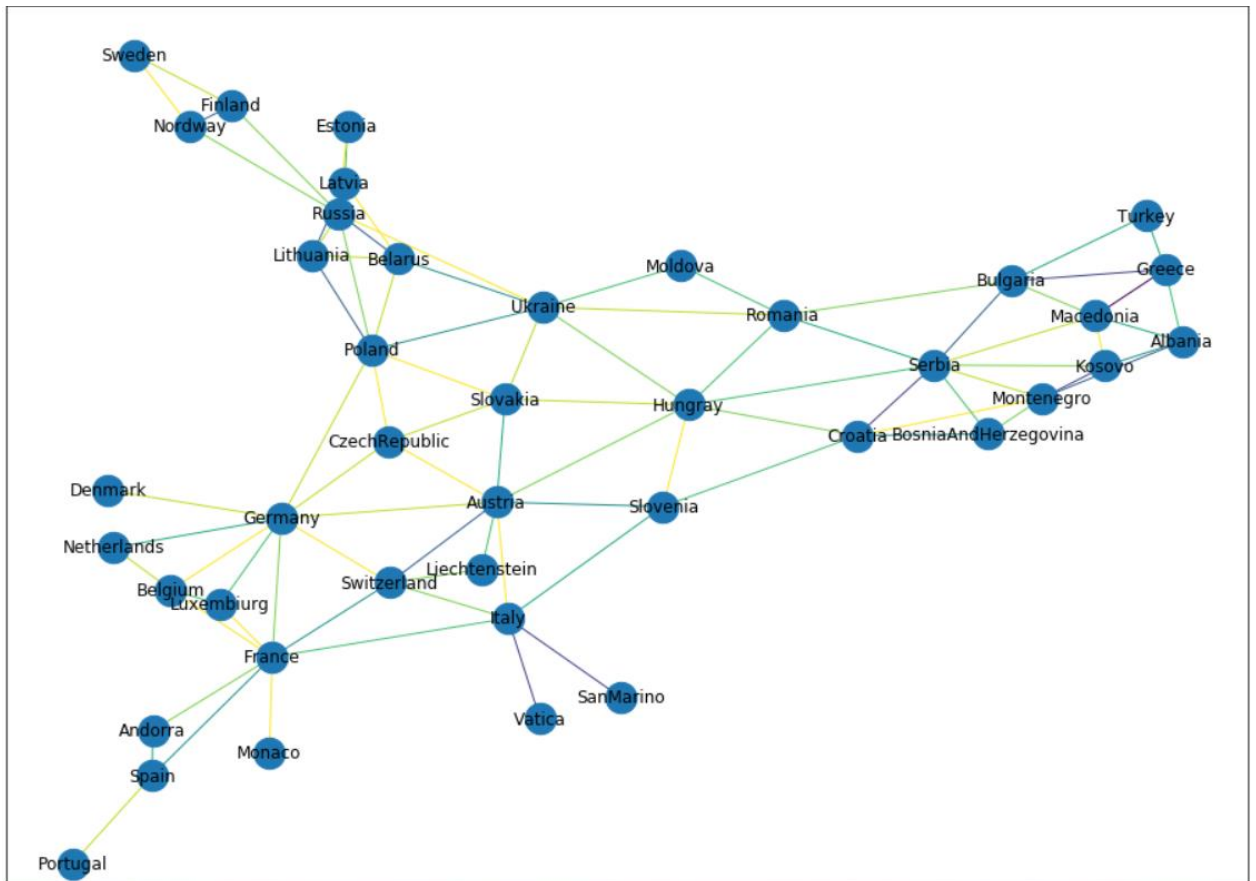


D : Find the minimum edge coloring $X : E \rightarrow N$ of G.

Рёберная раскраска – никакие два смежных ребра не имеют один и тот же цвет. Вершина “Germany” имеет степень равную 9, сл-но, данная раскраска имеет больше или равно 9 цветов.

```
plt.figure(figsize=(16, 12))
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'), node_size=500)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos, edge_color=colors_map)

plt.show()
```



Цвета не особо различимы, но вот сам colors_map:

```
print(colors_map)
```

```
[10, 9, 8, 7, 6, 5, 4, 10, 9, 10, 9, 10, 9, 8, 7, 6, 9, 10, 8, 7, 9, 10, 8, 7, 8, 10, 9, 7, 6, 5, 8, 7, 6, 5, 4, 2, 0, 6, 8, 1, 0, 9, 8, 6, 10, 5, 10, 7, 10, 9, 10, 9, 9, 4, 5, 7, 4, 10, 4, 9, 8, 5, 8, 10, 10, 8, 7, 9, 7, 9, 8, 6, 4, 7, 6, 5, 10, 2, 10, 4, 9, 8, 7, 2, 8, 9, 2, 2]
```

E: Find the maximum clique $Q \subseteq V$ of G .

Кликкой неориентированного графа называется подмножество его вершин, любые две из которых соединены ребром.

Максимальная клика — это клика, которая не может быть расширена путём включения дополнительных смежных вершин, то есть нет клики большего размера, включающей все вершины данной клики. Наибольшая клика — это клика максимального размера для данного графа.

```

cliques = [clique for clique in nx.find_cliques(G) if len(clique) > 3]
c = 0
for clique in cliques:
    c += 1
    print("Clique ", c, clique)

```

```

Clique 1 ['Serbia', 'Montenegro', 'BosniaAndHerzegovina', 'Croatia']
Clique 2 ['Belarus', 'Russia', 'Latvia', 'Lithuania']
Clique 3 ['Belarus', 'Russia', 'Poland', 'Ukraine']
Clique 4 ['Belarus', 'Russia', 'Poland', 'Lithuania']
Clique 5 ['Germany', 'Belgium', 'France', 'Luxemburg']

```

5 клик из 4 вершин. Из условия в “cliques” видно, что нет ни одной клики, где больше четырёх вершин.

F: Find the maximum stable set $S \subseteq V$ of G .

Независимое множество вершин (известное также как внутренне устойчивое множество) есть множество вершин графа G , такое, что любые две вершины в нем не смежны (никакая пара вершин не соединена ребром).

Независимое множество называется максимальным, когда нет другого независимого множества, в которое оно бы входило.

$S = \{\text{Albania, Czech Republic, Denmark, Estonia, Finland, Lithuania, Luxembourg, Monaco, Netherlands, San Marino, Serbia, Slovenia, Spain, Switzerland, Turkey, Ukraine, Vatican}\}$

Используем BFS.

```

def check(first):
    global matrix, friends, colors
    q = [first]
    q1 = []
    while q:
        x = q.pop()
        if friends[x] == 0:
            colors[x] = 1
        else:
            colors[x] = 0

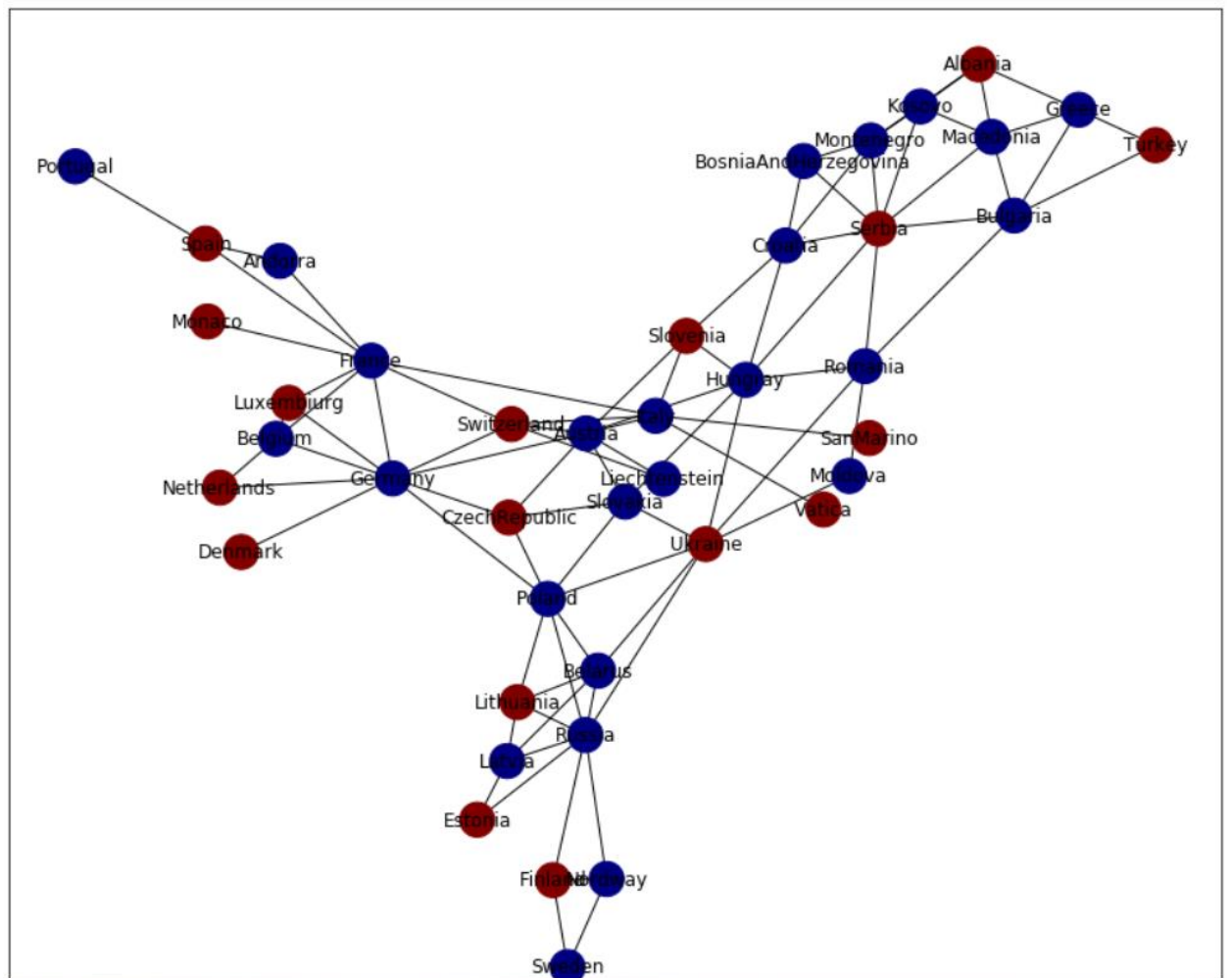
        for v in matrix[x]:
            if colors[v] == -1:
                friends[v] = max(friends[v], colors[x])
                q1.append(v)

    if not q:
        q = q1[:]
        q1 = []

```

Albania, CzechRepublic, Denmark, Estonia, Finland, Lithuania, Luxembiurg, Monaco, Netherlands, SanMarino, Serbia, Slovenia, Spain, Switzerland, Turkey, Ukraine, Vatica,

Вершины из S окрашены в красный цвет:



Паросочетание в графе - произвольное множество рёбер двудольного графа такое, что никакие два ребра не имеют общей вершины

Максимальное паросочетание - паросочетание, которое не содержится ни в каком другом паросочетании этого графа.

Наибольшее паросочетание – максимальное паросочетание, содержащее в себе максимальное число рёбер.

```
answer = nx.maximal_matching(G)

for i in answer:
    print(i)
print(len(answer))

('Liechtenstein', 'Switzerland')
('Slovakia', 'Poland')
('Hungary', 'Croatia')
('Belarus', 'Latvia')
('Germany', 'Belgium')
('Lithuania', 'Russia')
('Slovenia', 'Italy')
('Albania', 'Greece')
('Macedonia', 'Bulgaria')
('Spain', 'Portugal')
('Norway', 'Sweden')
('BosniaAndHerzegovina', 'Montenegro')
('Andorra', 'France')
('Austria', 'CzechRepublic')
('Romania', 'Serbia')
('Ukraine', 'Moldova')
16
```

H: Find the minimum vertex cover $R \subseteq V$ of G .

Вершинным покрытием графа G называется такое множество V его вершин, что у любого ребра в G хотя бы одна из вершин лежит в V .

Вершинное покрытие R графа G является минимальным вершинным покрытием графа G , если не существует такого множества $U \subseteq V$, что $U \subseteq R$

Минимальное вершинное покрытие R графа G является наименьшим вершинным покрытием графа G , если оно содержит минимальное количество вершин.

Th: Множество независимо тогда и только тогда, когда его дополнение является вершинным покрытием.

Consequence: Независимое множество является наибольшим тогда и только тогда, когда его дополнение является наименьшим вершинным покрытием.

Следовательно, наименьшим вершинным покрытием R является дополнение множества S (maximum stable set):

$R = \{\text{Albania, Czech Republic, Denmark, Estonia, Finland, Lithuania, Luxembourg, Monaco, Netherlands, San Marino, Serbia, Slovenia, Spain, Switzerland, Turkey, Ukraine, Vatican}\}$

I : Find the minimum edge cover $F \subseteq E$ of G .

Рёберное покрытие графа — это множество рёбер F , такое, что каждая вершина графа инцидентна по меньшей мере одному ребру из F .

Рёберное покрытие F графа G является минимальным реберным покрытием графа G , если не существует такого множества $K \subseteq E$, что $K \subseteq F$

Минимальное рёберное покрытие F графа G является наименьшим реберным покрытием графа G , если оно содержит минимальное количество рёбер.

Алгоритм: найдём наибольшее паросочетание в графе и включим в него рёбра, необходимые для покрытия непокрытых паросочетанием вершин.

В пункте (g)(Maximum matching) мы нашли наибольшее паросочетание, но оно не покрыло вершины : Denmark, Estonia, Finland, Kosovo, Luxembourg, Monaco, Netherlands, San Marino, Turkey, Vatican

Добавим рёбра множеству F :

Denmark – Germany, Estonia – Latvia, Finland – Sweden, Kosovo – Serbia, Luxembourg – France, Monaco – France, Netherlands – Belgium, San Marino – Italy, Turkey – Greece, Vatican – Italy

Множество F является наименьшим реберным покрытием графа G :

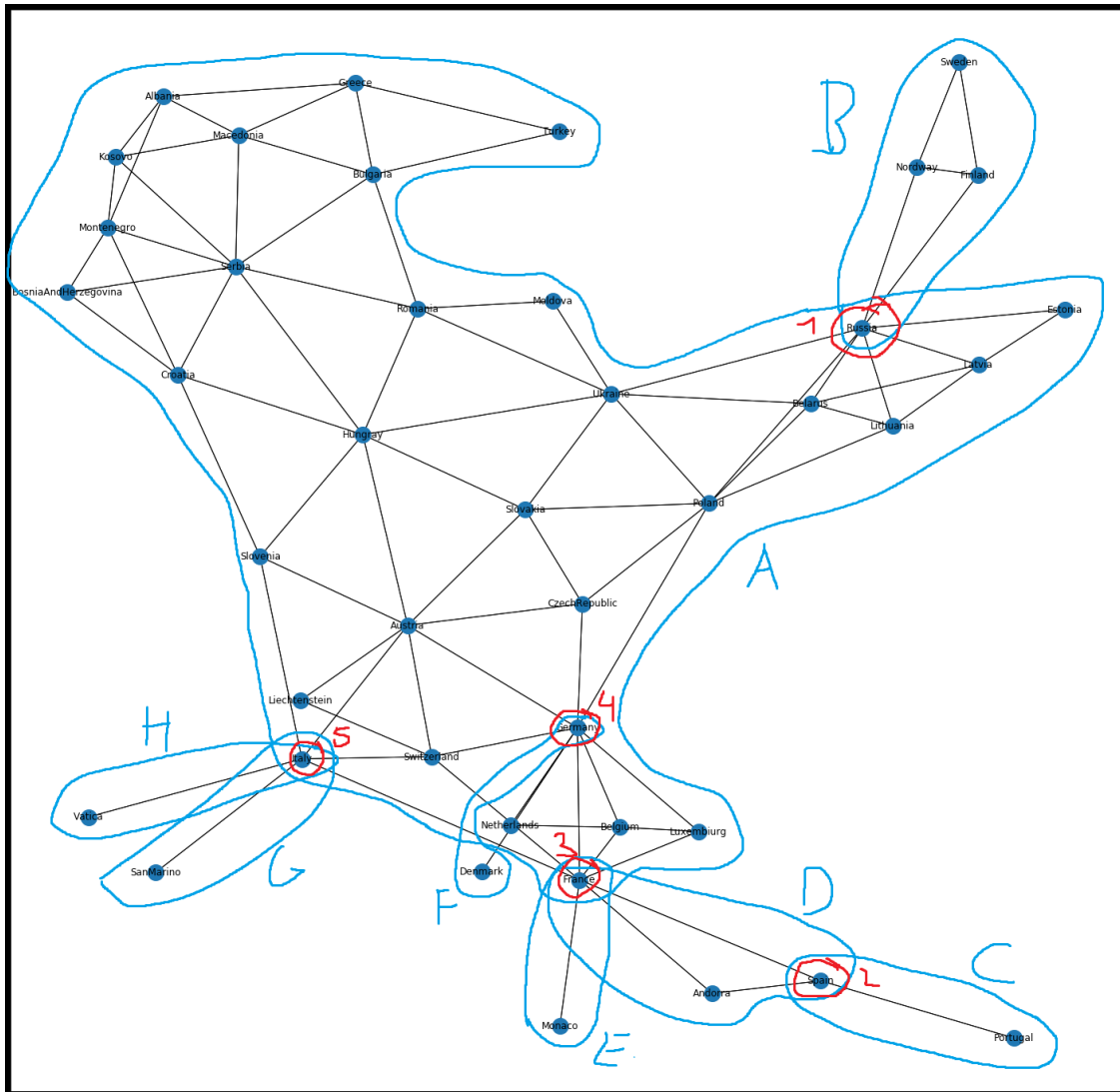
F = Hungary – Croatia, Liechtenstein – Switzerland, Albania – Greece, Slovenia – Italy, Andorra – France, Belarus – Latvia, Norway – Sweden, Ukraine – Moldova, Denmark – Germany, Estonia – Latvia, Finland – Sweden, Kosovo – Serbia, Luxembourg – France, Monaco – France, Netherlands – Belgium, San Marino – Italy, Turkey – Greece, Vatican – Italy, Slovakia – Poland, Lithuania – Russia, Austria – Czech Republic, Spain – Portugal, Germany – Belgium, Macedonia – Bulgaria, Romania – Serbia, Bosnia And Herzegovina – Montenegro

```
plt.figure(figsize=(18, 14))
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'), node_size=500)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos, edge_color=colors)

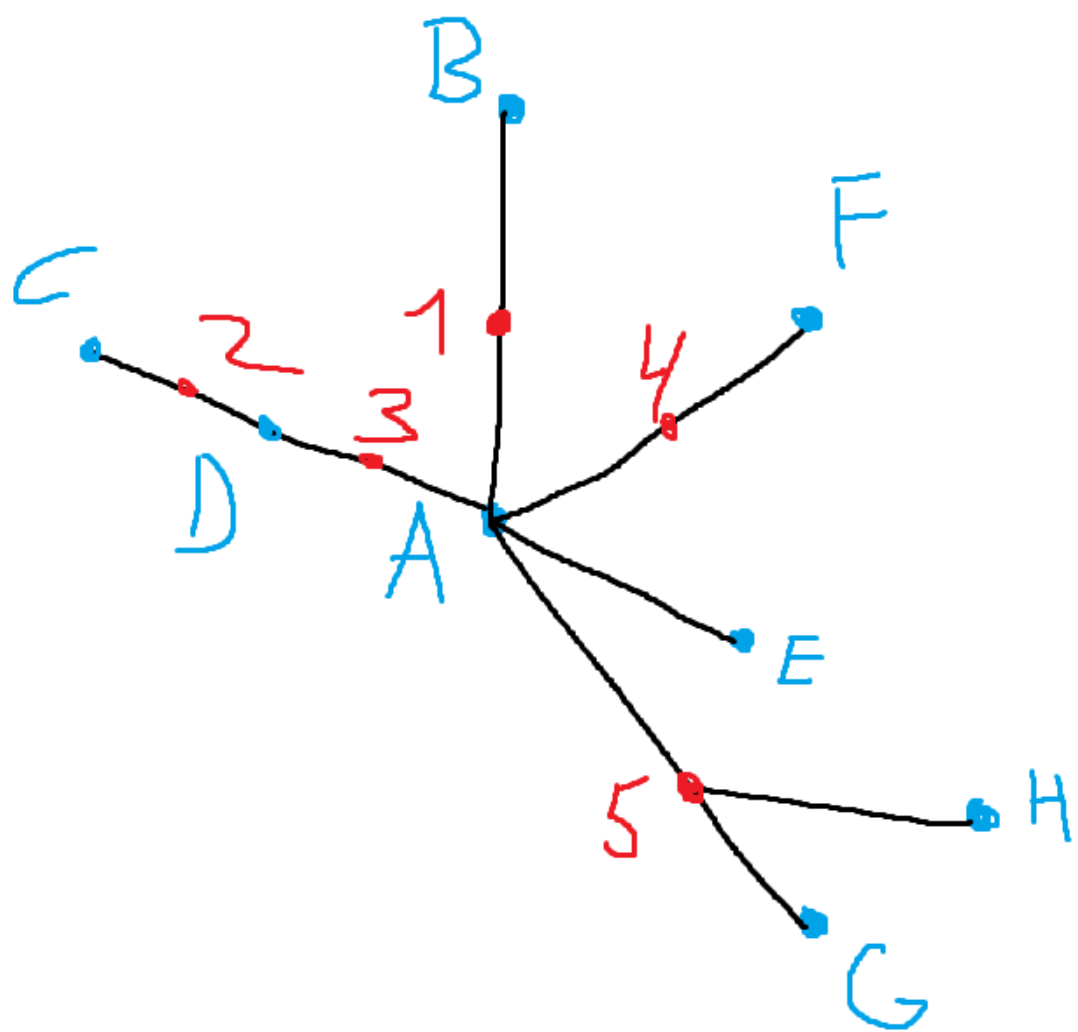
plt.show()
```

Снизу изображён граф, где рёбра из F окрашены в красный цвет:

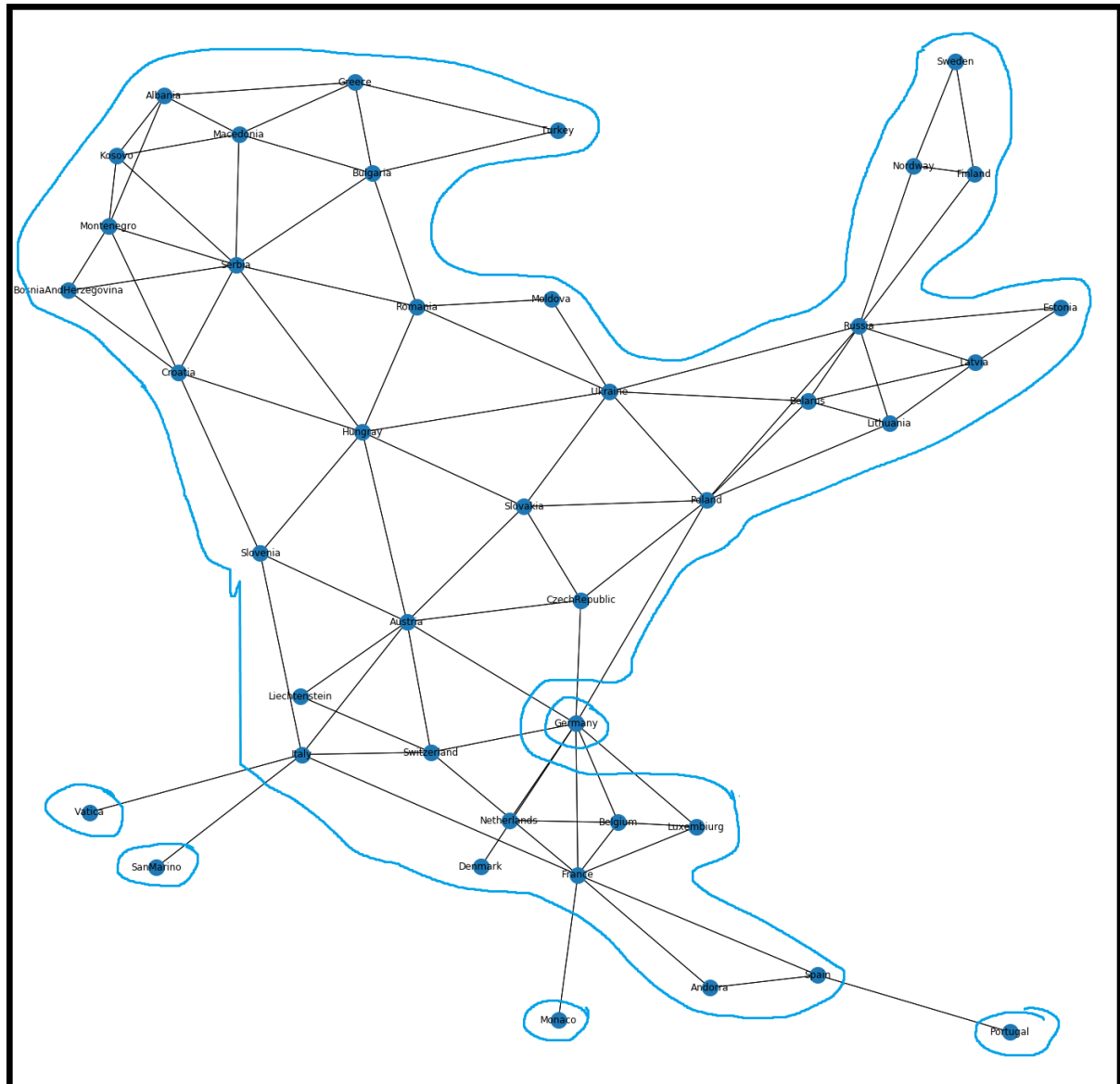
L: Find all 2-vertex-connected components (blocks) and draw a block-cut tree of G^* .



Блоки – синий цвет, шарниры – красный цвет.



M : Find all 2-edge-connected components of G^* .

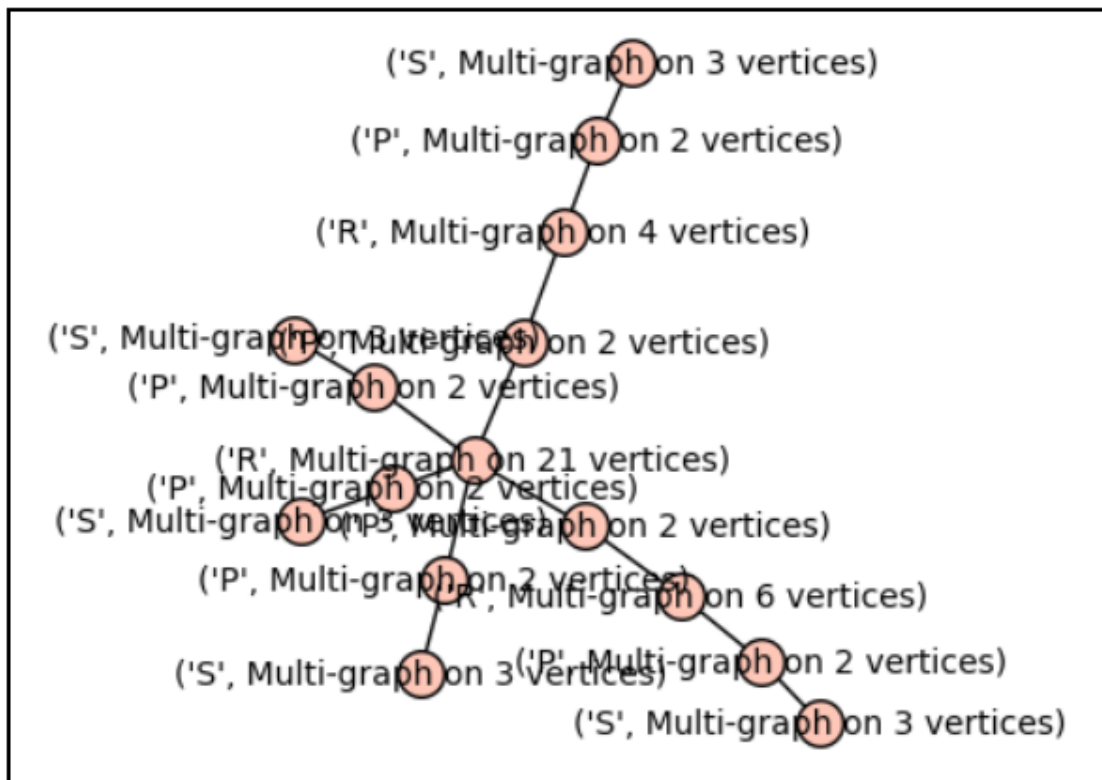


N: Construct an SPQR tree of the largest biconnected component of G .

Трёхсвязные компоненты двусвязного графа — это система более мелких графов, описывающих все 2-вершинные сечения графа. SPQR-дерево – это древесная структура данных для представления трёхсвязных компонент графа.

```

1 from sage.graphs.connectivity import TriconnectivitySPQR
2 from sage.graphs.connectivity import spqr_tree_to_graph
3 G = Graph ([[2, 8], [2, 13], [2, 15], [2, 19], [2, 34], [2, 35],
4 [2, 38], [0, 14], [0, 22], [3, 18], [3, 20], [3, 28],
5 [3, 31], [3, 40], [4, 12], [4, 13], [4, 21], [4, 26],
6
7 [6, 14], [6, 22], [6, 30], [6, 39], [22, 14], [15, 7],
8
9 [15, 30], [15, 34], [15, 35], [15, 40], [13, 8], [13, 12],
10
11 [13, 21], [13, 26], [13, 28], [13, 38], [14, 39], [18, 10],
12
13 [18, 20], [18, 31], [20, 28], [20, 31], [19, 38], [21, 12],
14
15 [23, 30], [23, 40], [28, 8], [28, 31], [28, 34], [28, 40],
16
17 [30, 40], [34, 40], [34, 8], [35, 7], [40, 31], [12, 38], [7, 5],
18 [10, 31], [17, 25], [17, 0], [17, 22], [17, 33], [25, 7], [25, 5],
19 [25, 33], [25, 0], [33, 15], [33, 30], [33, 6], [33, 22],
20 [33, 5], [33, 7], [16, 12], [16, 35], [16, 38], [2, 16]])
21
22 tric = TriconnectivitySPQR(G)
23 T = tric.get_spqr_tree()
24 T.show()
```



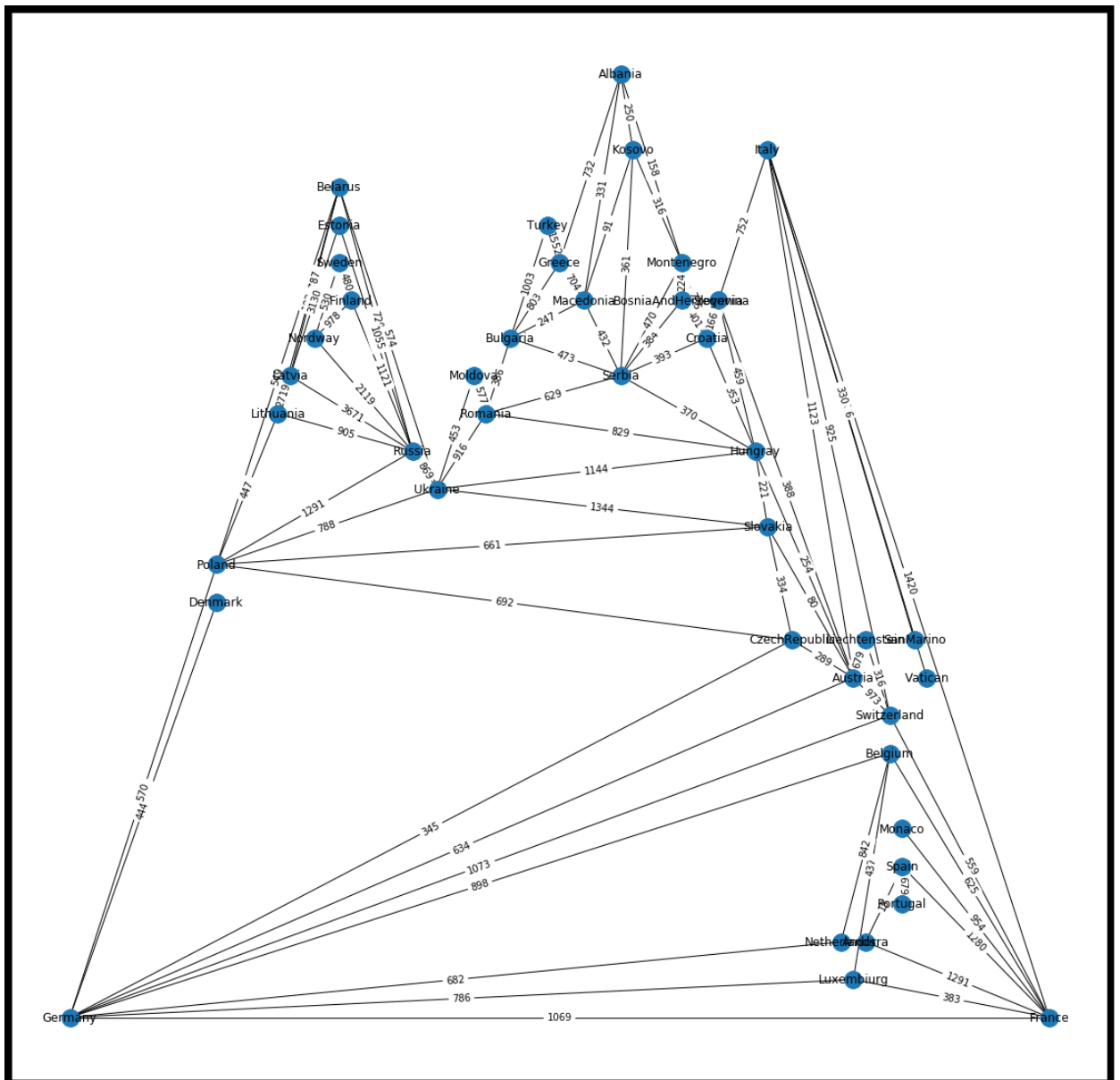
o: Add the weight function $w : E \rightarrow \mathbb{R}$ denoting the distances between capitals. Find the minimum (w.r.t. the total weight of edges) spanning tree T for the maximum connected component of the weighted Europe graph $G_w = (V, E, w)$.

Graph with a weight function:

```

G.add_weighted_edges_from(edges)
weights = nx.get_edge_attributes(G, 'weight')
plt.figure(figsize=(20, 20))
pos = nx.planar_layout(G)
nx.draw_networkx(G, pos=pos)
nx.draw_networkx_nodes(G, pos, node_size=300)
nx.draw_networkx_edge_labels(G, pos, edge_labels=weights)
plt.show()

```



Минимальное остовное дерево, полученное с помощью алгоритма Прима:

```
def prim_mst(n, m, edges):
    def find(v):
        nonlocal p
        while (v != p[v]):
            v = p[v]
        return v

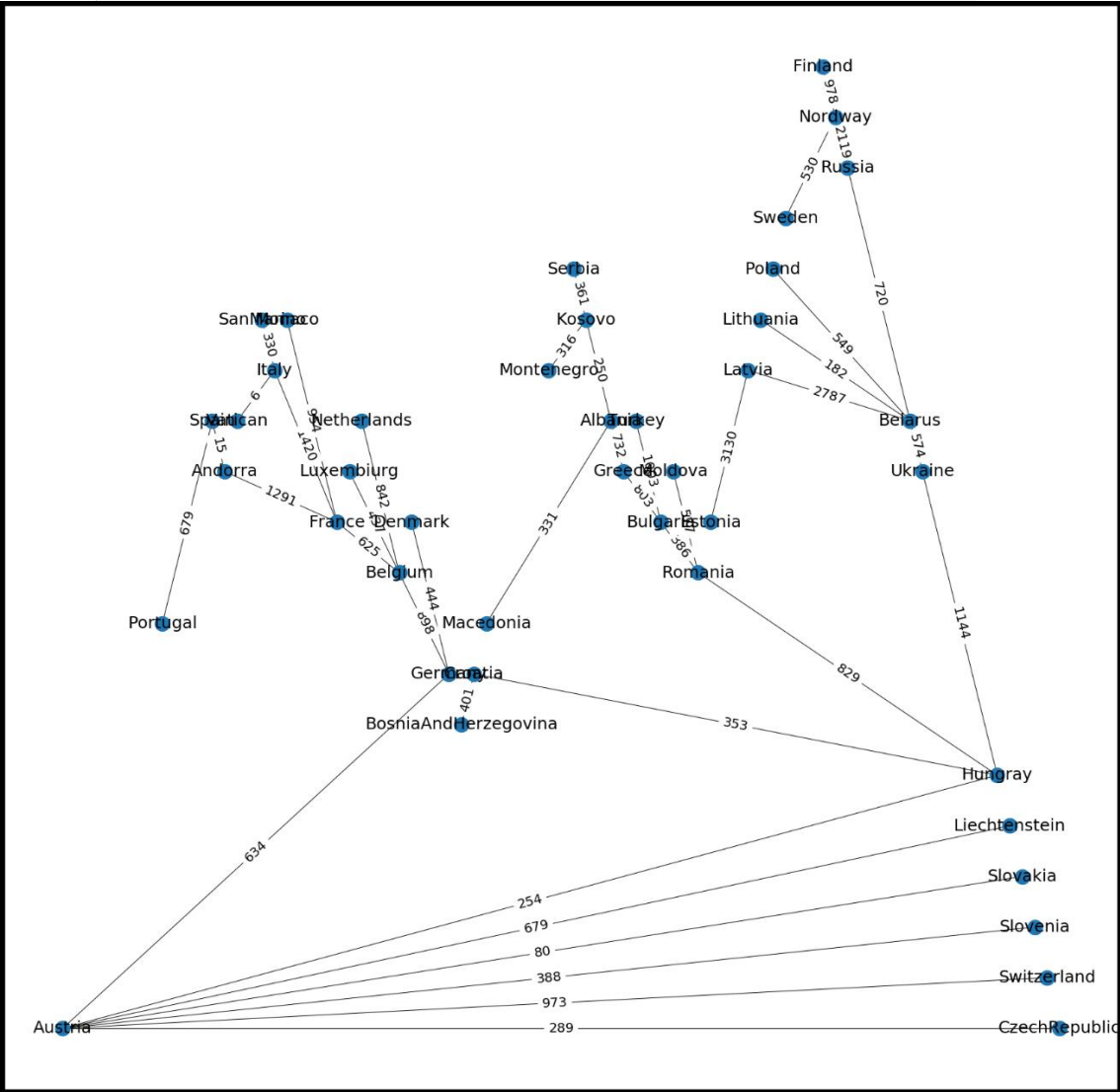
    def check(v1, v2):
        return find(v1) == find(v2)

    def union():
        nonlocal p, rank, x, y, z, ans
        v1 = find(x)
        v2 = find(y)
        if rank[v1] > rank[v2]:
            v1, v2 = v2, v1
        rank[v2] += rank[v1]
        p[v1] = v2

    p = [i for i in range(n)]
    rank = [1] * n
    ans = 0
    ans_edges = []
    for i in range(m):
        x, y, z = edges[i]
        if not check(x, y):
            union()
            ans += z
            ans_edges.append([country[x], country[y], z])

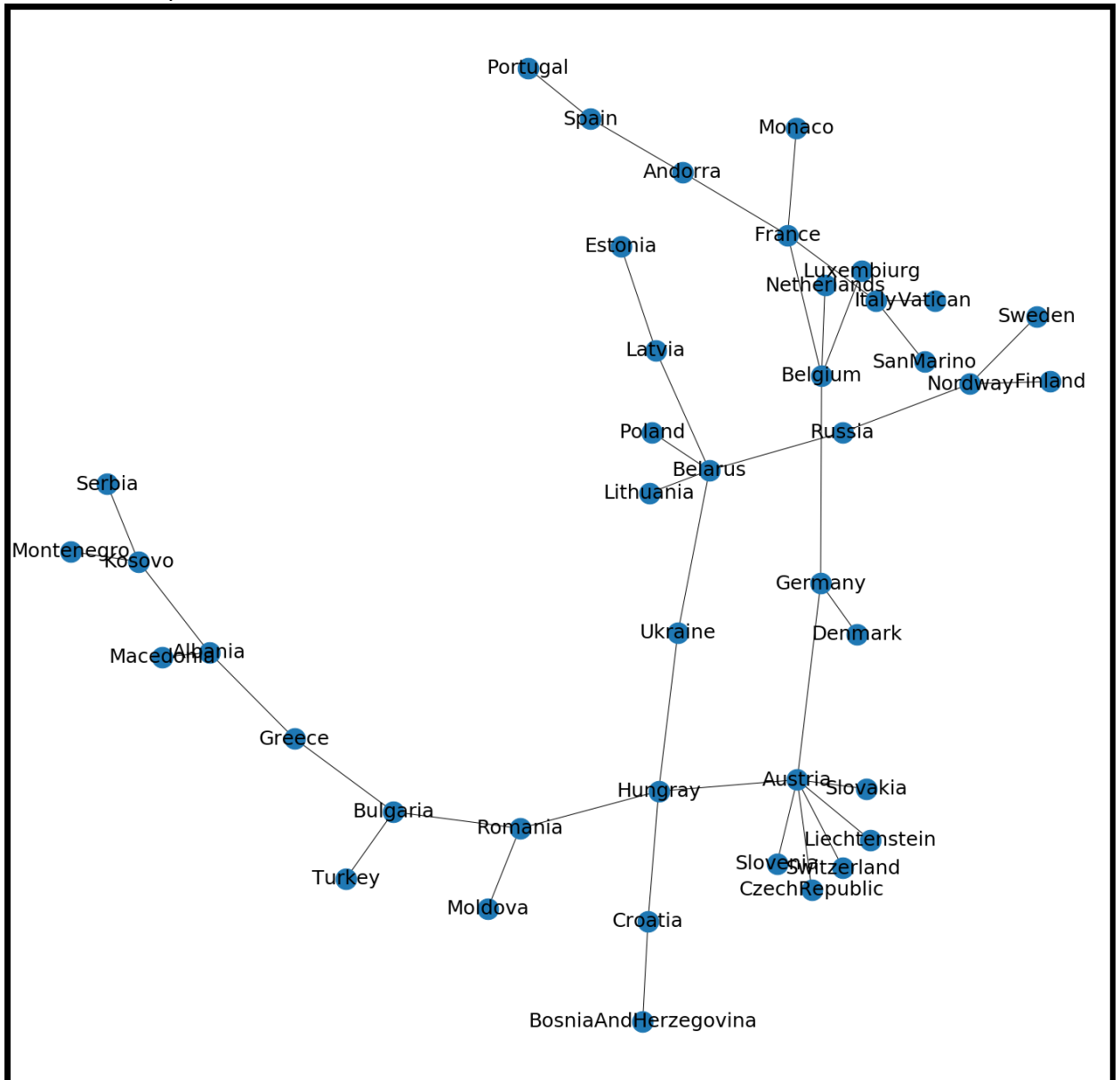
    print('Weight of MST:', ans)

# Plotting MST
G = nx.Graph()
G.add_weighted_edges_from(ans_edges)
weights = nx.get_edge_attributes(G, 'weight')
plt.figure(figsize=(30, 30))
pos = nx.planar_layout(G)
nx.draw_networkx(G, pos=pos, font_size=25)
nx.draw_networkx_nodes(G, pos, node_size=500)
nx.draw_networkx_edge_labels(G, pos, edge_labels=weights, font_size=20)
plt.show()
```



P: Find $\text{centroid}(T)$ (w.r.t. the edge weight function w).

Центроидом дерева называется такая вершина v дерева t , после удаления которой дерево разбивается на несколько k поддеревьев t_1, t_2, \dots, t_k , таких что для каждого i : $|t_i| \leq n/2$, то есть размер каждого поддерева не превосходит половины размера исходного дерева.



Центроидом является вершина “Hungary”

Q: Construct the Prufer code for T .

Код Прюфера — это способ взаимно однозначного кодирования помеченных деревьев с n вершинами с помощью последовательности $n - 2$ целых чисел в отрезке $[1; n]$. Иными словами, код Прюфера — это **биекция** между всеми остовными деревьями полного графа и числовыми последовательностями.

```

G_sec = nx.Graph()
G_sec.add_edges_from(answer_edges)
prufer_code = nx.to_prufer_sequence(G_sec)

for i in prufer_code:
    print(country[i], end = ', ')|

prim_mst(V, E, edges_name)

```

Croatia, Hungray, Austria, Germany, Latvia, Nordway, Belarus, Austria, Belarus, Belgium, Albania, Romania, France, Kosovo, Belgium, Belarus, Spain, Italy, Kosovo, Albania, Greece, Bulgaria, Austria, Austria, Andorra, France, Nordway, Russia, Belarus, Ukraine, Austria, Bulgaria, Romania, Hungray, Hungray, Austria, Germany, Belgium, France, Italy,

Задание №2.

Theorem 1. (Triangle Inequality) For any connected graph $G = \langle V, E \rangle$:

$\forall x, y, z \in V : \text{dist}(x, y) + \text{dist}(y, z) \geq \text{dist}(x, z)$

Док-во:

Воспользуемся методом “от противного”. Пусть $\text{dist}(x, y) + \text{dist}(y, z) < \text{dist}(x, z)$.

Следовательно, сумма длин кратчайших путей между вершинами x и y , и между вершинами y и z меньше, чем длина кратчайшего пути между вершинами x и z . Но $\text{dist}(x, z)$ – путь наименьше длины между вершинами x и z . Мы получаем противоречие: пути между двумя вершинами, длина которого меньше длины кратчайшего пути между ними не может быть, сл-но, Triangle Inequality for any connected graph справедливо.

Theorem 2. For any connected graph G : $\text{rad}(G) \leq \text{diam}(G) \leq 2 \text{rad}(G)$.

Док-во:

Т.к. радиус – это минимальный эксцентриситет, а диаметр – максимальный, то очевиден факт, что радиус не может быть больше диаметра.

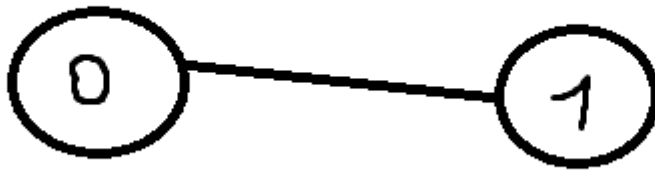
Пусть u и v такие вершины графа, что $\text{dist}(u, v) = \text{diam}(G)$. Пусть c – центральная вершина, так что $e(c) = \text{rad}(G)$. Значит, что ни одна вершина не находится на расстоянии большему, чем расстояние, равное радиусу графа, от вершины c . В частности, $\text{dist}(u, c)$ и $\text{dist}(v, c)$ оба меньше или равны $\text{rad}(G)$. Сл-но, $\text{dist}(u, c) + \text{dist}(v, c) \leq 2\text{rad}(G)$. По неравенству треугольника, $\text{dist}(u, v) \leq \text{dist}(u, c) + \text{dist}(v, c)$. Из этого следует, что $\text{rad}(G) \leq \text{diam}(G) \leq 2\text{rad}(G)$ ч.т.д.

Theorem 3. A connected graph $G = \langle V, E \rangle$ is a tree (i.e. acyclic graph) iff $|E| = |V| - 1$

Док-во:

- Докажем, что связный граф $G = \langle V, E \rangle$, в котором $|E| = |V| - 1$, является деревом. (док-во влево)**

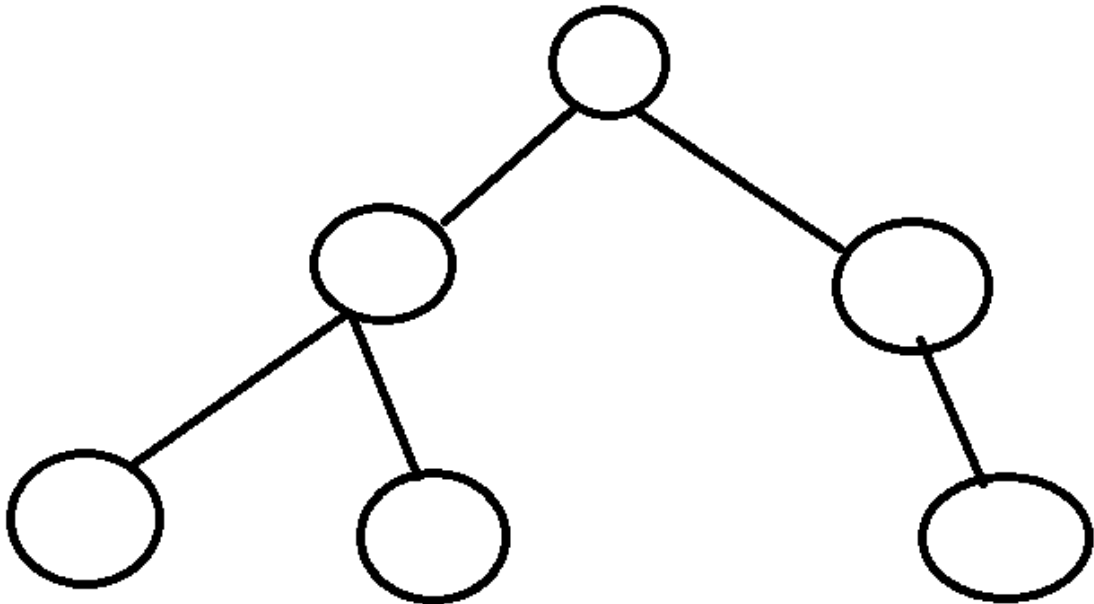
Представим граф, состоящий из одной вершины и не имеющий рёбер. То есть, $|E| = 0$, $|V| = 1$. Добавим вершину и ребро, которое будет инцидентно изначальной вершине и новой добавленной. Повторяя эту операцию несколько раз, мы увидим, что для получившегося графа после каждой итерации будет справедливо $|E| = |V| - 1$. Связность нарушаться не будет, тк мы всё время делали действия с данной компонентной связности. Проверим граф на ацикличность. Рассмотрим первую итерацию:



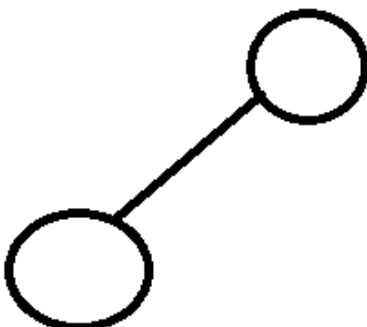
Этот граф ацикличен, тк между любыми двумя вершинами существует только одна простая цепь. На следующий итерациях, добавляя вершину, она будет смежна вершине из ацикличного графа на картине, то есть, такой граф тоже будет ацикличен. Сл-но, делаем вывод, что граф $G = \langle V, E \rangle$ является деревом(связность и ацикличность).

2. Докажем, что если граф $G = \langle V, E \rangle$ является связным, то $|E| = |V| - 1$ (док-во вправо)

Рассмотрим дерево:



Удалим из него какой-нибудь лист и ребро, инцендентное ему. Лист – вершина со степенью, равной единице, сл-но, будет удалено только одно ребро. Повторяя эту операцию, граф так же будет оставаться деревом. После $(N - 2)$ итераций, граф будет выглядеть так:



После следующей итерации останется граф, состоящий только из одной вершины и не имеющий рёбер. То есть, после $(N - 2 + 1)$ итерацию будет удалено $(N - 1)$ вершин и рёбер, сл-но, изначальный граф имел N вершин и $(N-1)$ рёбер, то есть $|E| = (N - 1)$, а $|V|=N$. Получаем, $|E| = |V| - 1$.

Доказали в обе стороны, сл-но, связный граф $G = \langle V, E \rangle$ - дерево, тогда и только тогда, когда, $|E| = |V| - 1$. Чтд.

Theorem 4. Given a connected graph G with n vertices, if $\delta(G) \geq \lfloor n/2 \rfloor$, then $\lambda(G) = \delta(G)$.

Док-во:

1. Пусть $\delta(G) > \lambda(G)$. Удалим $\lambda(G)$ рёбер получим, что для каждой вершины u из V останется как минимум $\delta(G) - \lambda(G) > 0$ соседей, то есть противоречие.
2. Пусть $\delta(G) < \lambda(G)$. Удалим $\lambda(G)$ рёбер получим, что для каждой вершины u из V останется как минимум $\delta(G) - \lambda(G) < 0$ соседей, то есть противоречие.

Следовательно, из п(1,2) : $\delta(G) = \lambda(G)$. Чтд.

Theorem 5. Every block of a block graph⁴ is a clique.

Док-во:

“По определению”

https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84

Блоковый граф (кликоее дерево) — вид неориентированного графа, в котором каждая компонента двусвязности (блок) является кликой.

Блоковые графы можно описать графами пересечений блоков произвольных неориентированных графов.