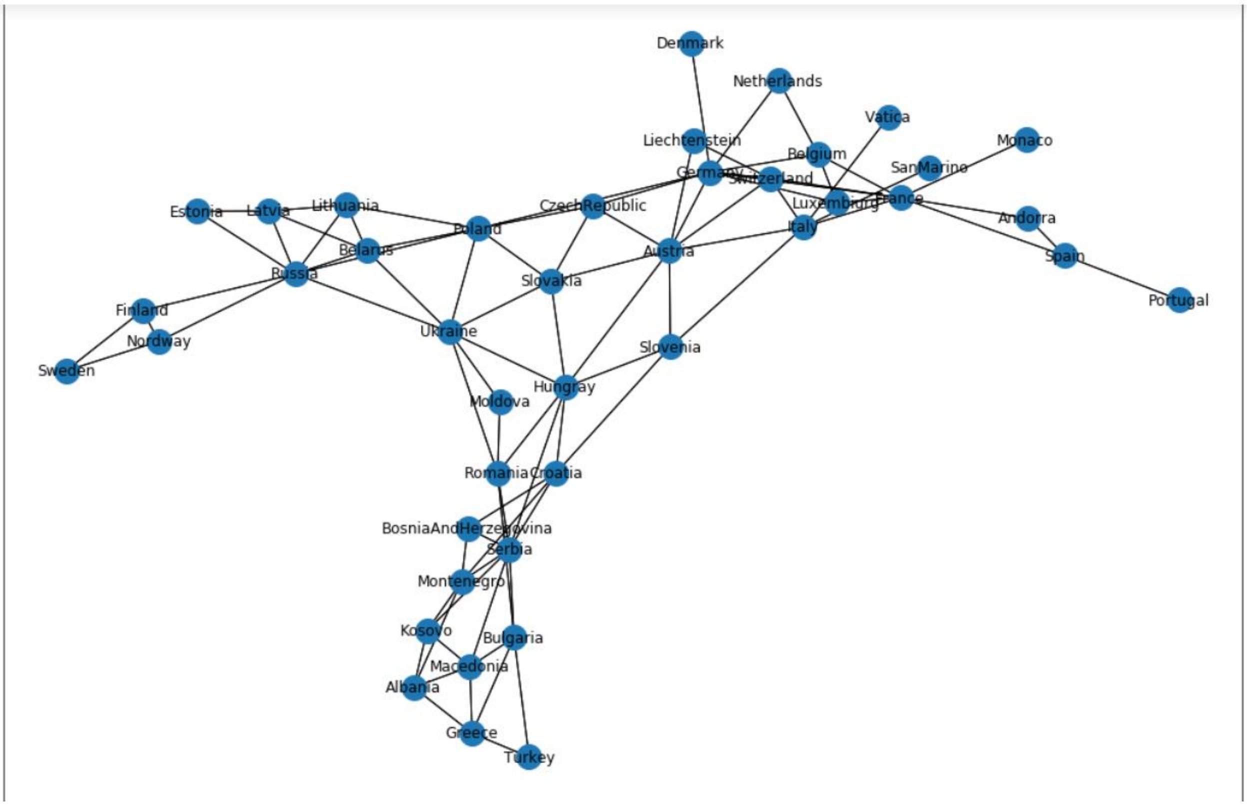
Homework #5
Graph Theory
Suslov Misha, M3104

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 рёбер 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 изображённых граф удовлетворяет определению плоского графа, т.е. условиям, сл-но является плоским.

B : Find $|V|$, $|E|$, δ(G), Δ(G), rad(G), diam(G), girth(G), center(G), κ(G), λ(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
```

δ (G) = 1

Δ(G) = 9

Радиус, диаметр, центр:

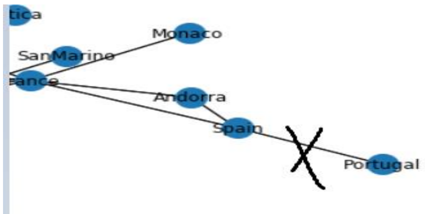
```
# Calculate the shortest distance using the Floyd-Warshall algorithm
distance = [[0] * V for i in range(V)]
inf = 10 ** 5
for i in range(V):
    for j in range(V):
        if matrix[i][j]:
            distance[i][j] = 1
        else:
            distance[i][j] = inf
for k in range(V):
    for i in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

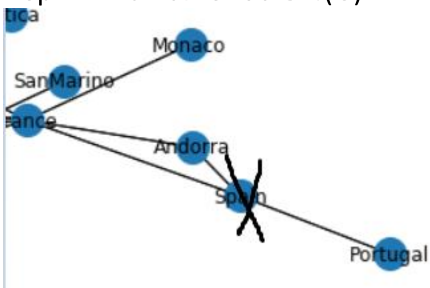
```
radius =  5
diameter =  8
center =  ['Austria', 'Belarus', 'Croatia', 'CzechRepublic', 'Germany', 'Hungray', 'Poland', 'Russia', 'Slovakia', 'Slovenia', 'Switzerland', 'Ukraine']
```

Рёберная связность λ(G):



Убираем ребро между Испанией и Португалией – получаем несвязный граф. λ(G) = 1

Вершинная связность κ(G):



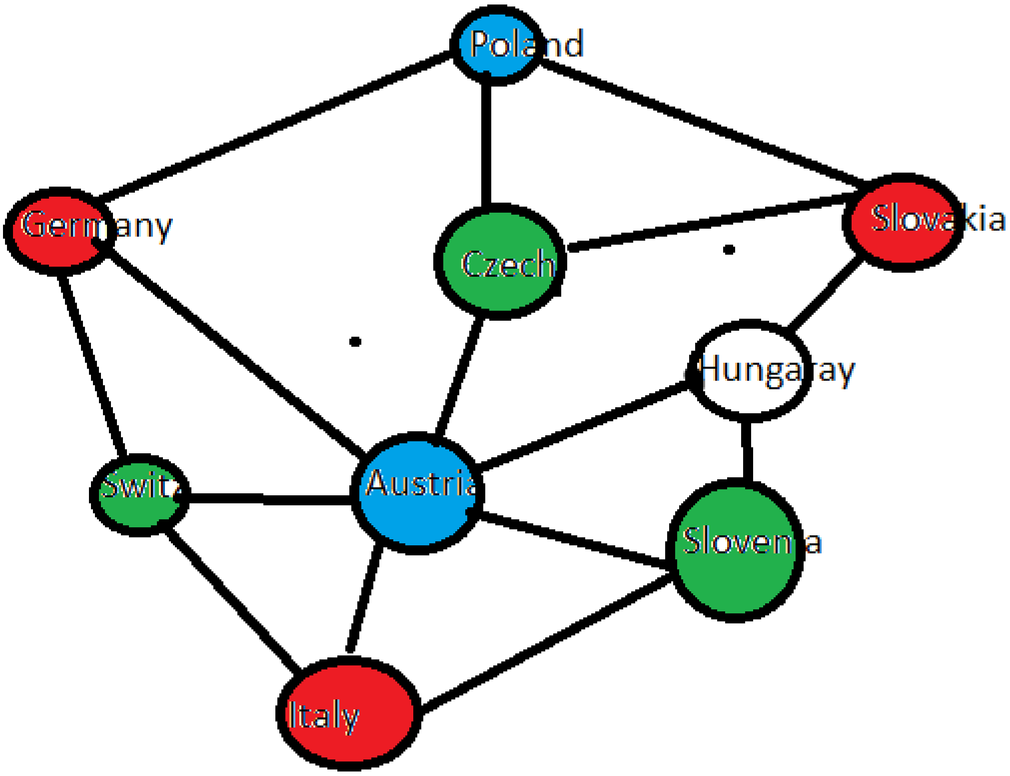
Убираем вершину с Испанией – получаем несвязный граф. κ(G) = 1

C: Find the minimum vertex coloring $Z : V \rightarrow$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.

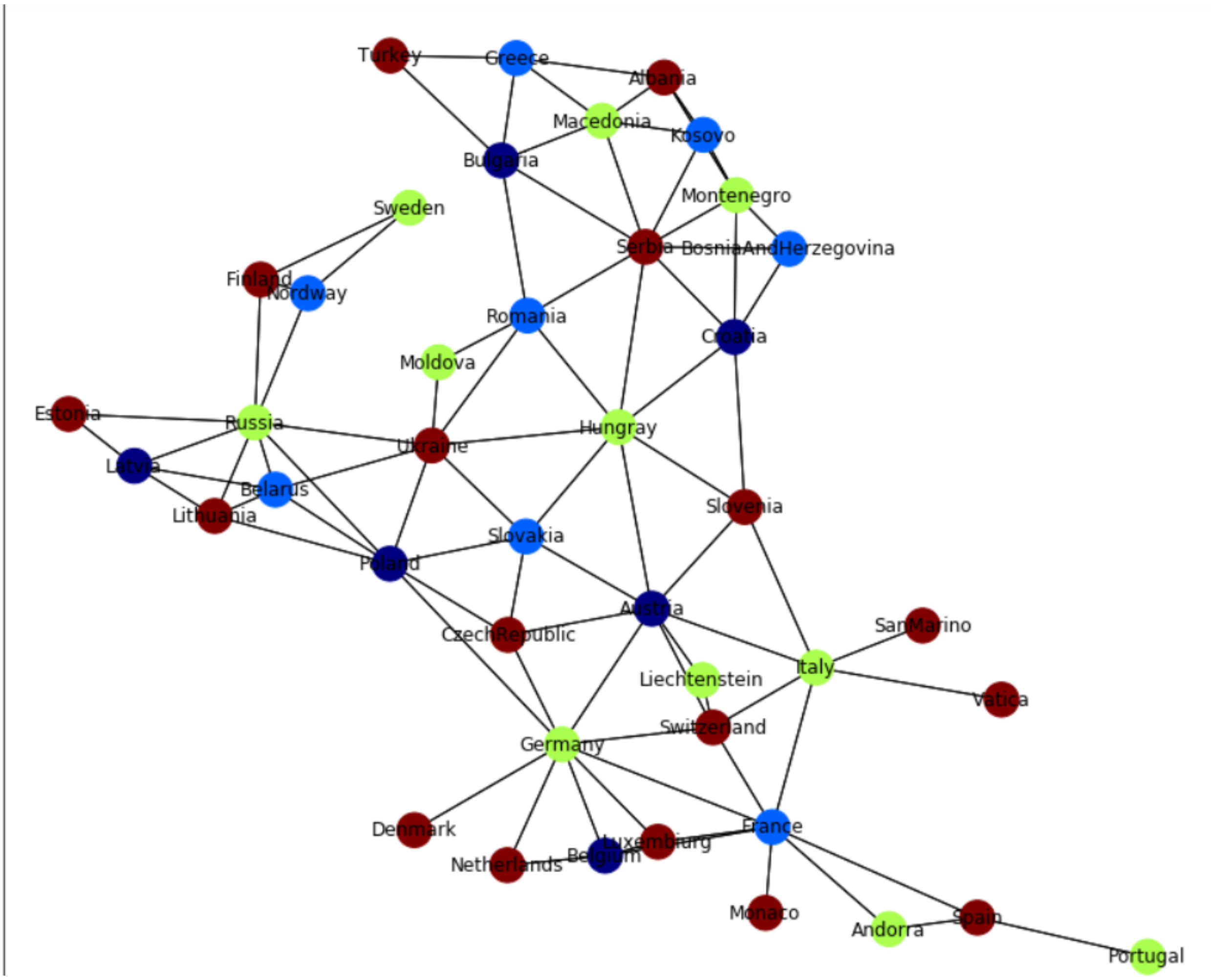
Докажем, что минимальное кол-во цветов будет равно 4:



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:

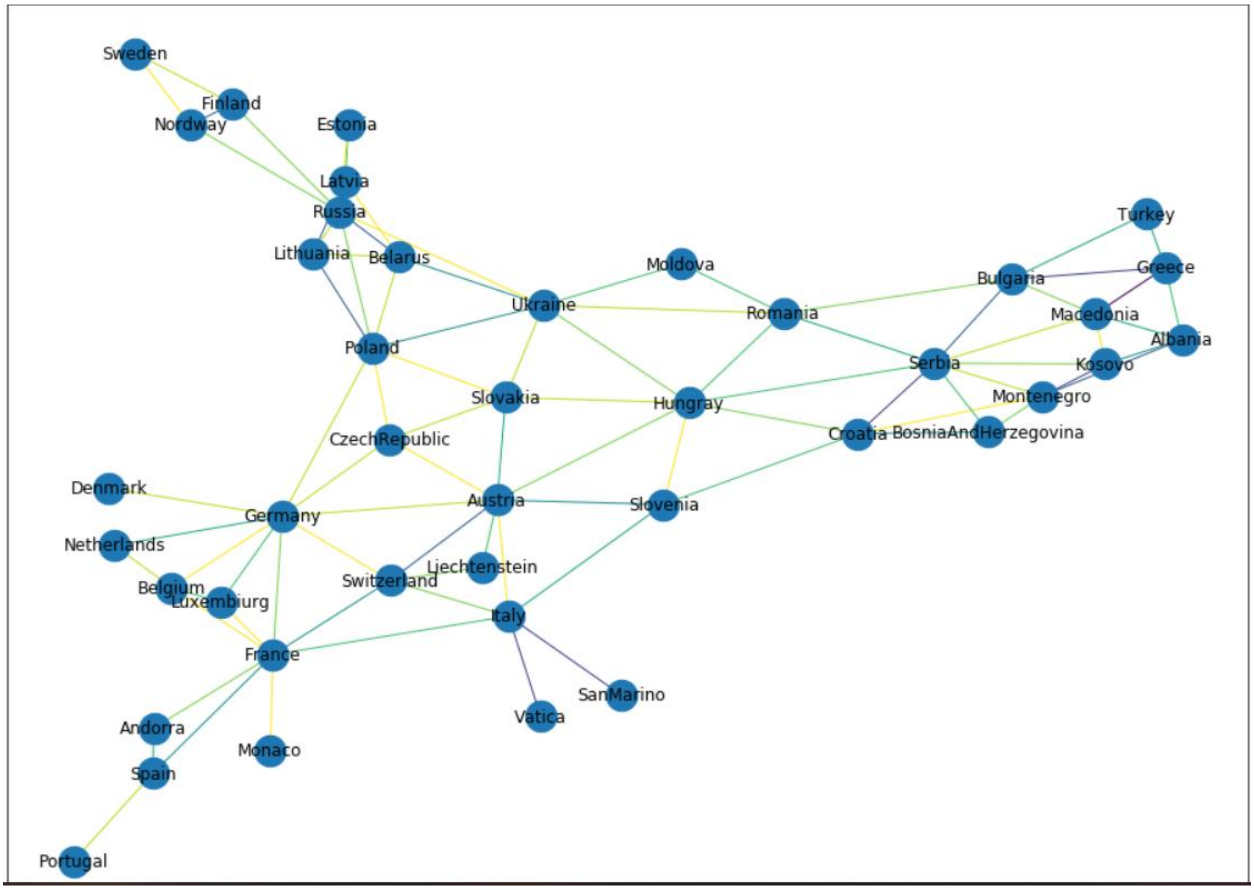
```
plt.figure(figsize=(14, 12))
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'), node_size=400, node_color=val)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_edges(G, pos, arrows=False)
plt.show()
```



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.

Кликой 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', 'Luxembiurg']
```

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

Используем BFS.

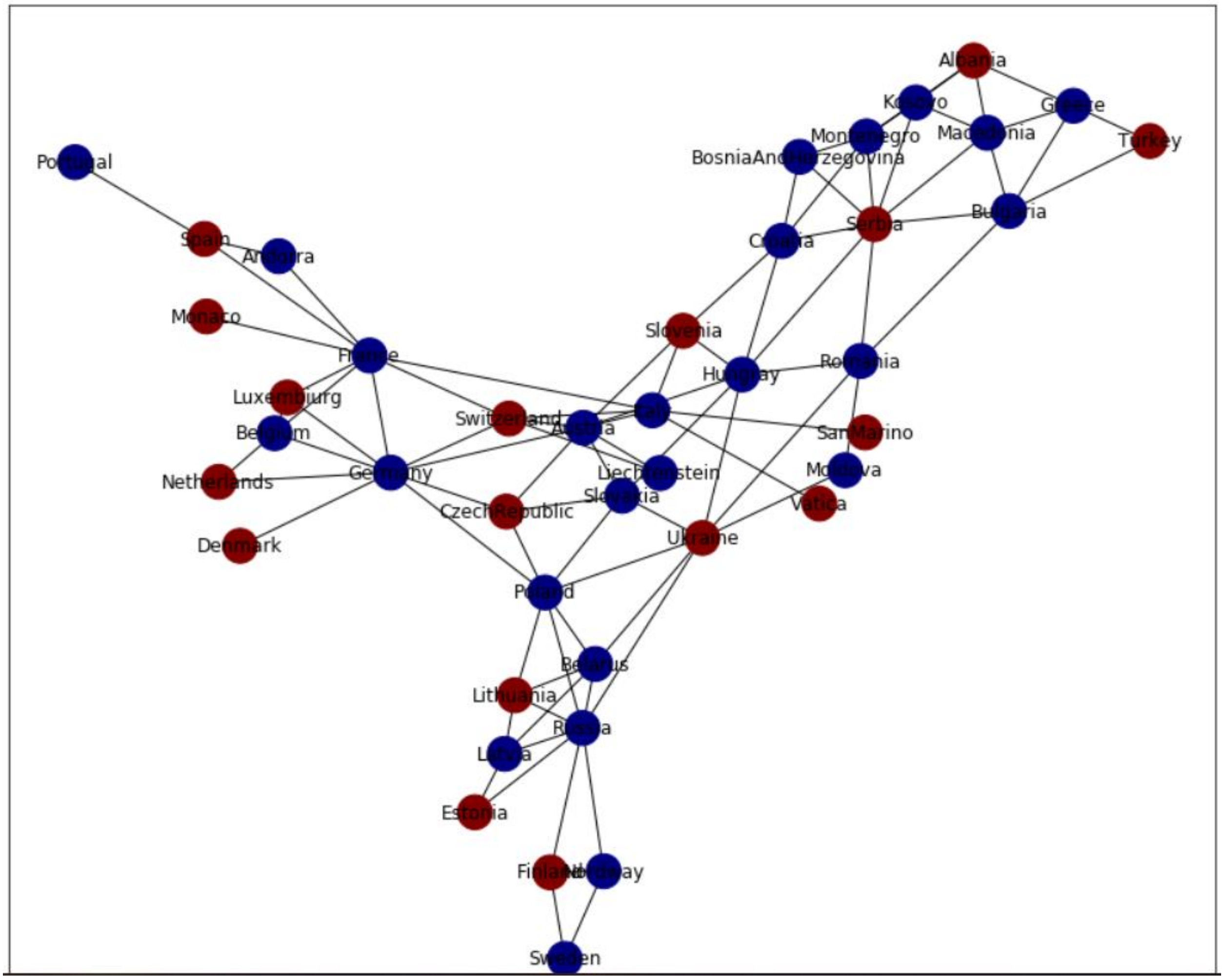
```
def check(first):
    global matrix, friends, colors
    q = [first]
    q1 = []
    while q:
        x = q.pop()
        if friends[x] == 0:
            colors[x] = 1
        else:
            colors[x] = 0

        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 = []
```

```
first = 0
check(first)
colors_map = dict()
c = 0
for i in colors:
    colors_map[country[c]] = i
    if i == 1:
        print(country[c], end=', ')
    c += 1

val = [colors_map.get(node, 0.25) for node in G.nodes()]
print()
```

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

```
In [10]: plt.figure(figsize=(14, 12))
         pos = nx.spring_layout(G)
         nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'), node_size=500, node_color=val)
         nx.draw_networkx_labels(G, pos)
         nx.draw_networkx_edges(G, pos)

         plt.show()
```

Вершины из S окрашены в красный цвет:



G: Find the maximum matching $M \subseteq E$ of G.

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 – макисимальное 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')
('Hungray', 'Croatia')
('Belarus', 'Latvia')
('Germany', 'Belgium')
('Lithuania', 'Russia')
('Slovenia', 'Italy')
('Albania', 'Greece')
('Macedonia', 'Bulgaria')
('Spain', 'Portugal')
('Nordway', '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 минимальных вершинным покрытием графа G, если не существует такого множества U⊆V, что U⊆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 = {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.

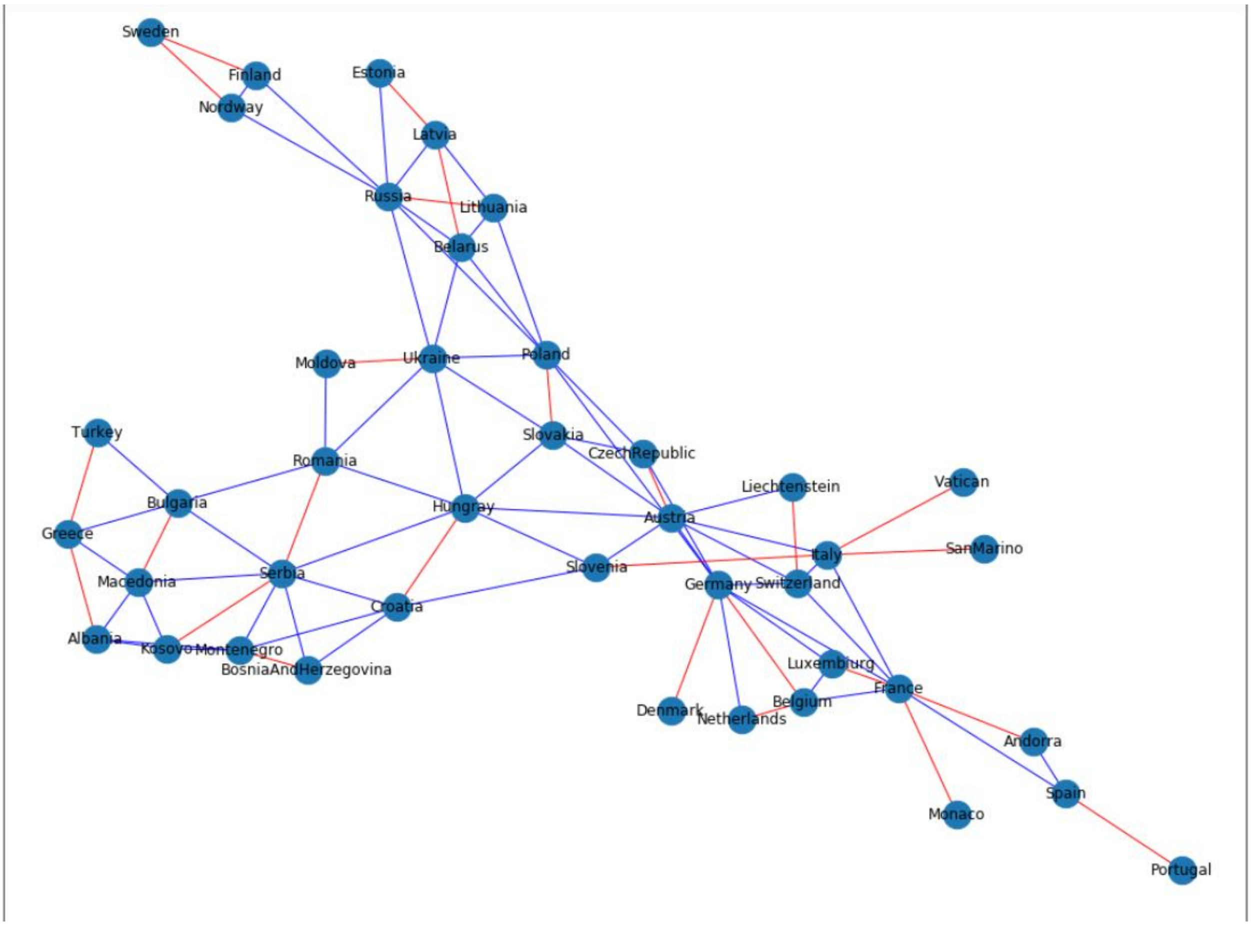
В пункте (g)(Maximum matching) мы нашли наибольшее паросочетание, но оно не покрыло вершины : Denmark, Estonia, Finland, Kosovo, Luxembourg, Monaco, Netherlands, San Marino, Turkey, Vatican

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:



J: Find the shortest closed path (circuit) W that visits every vertex of G

K: Find the shortest closed path (circuit) U that visits every edge of G.

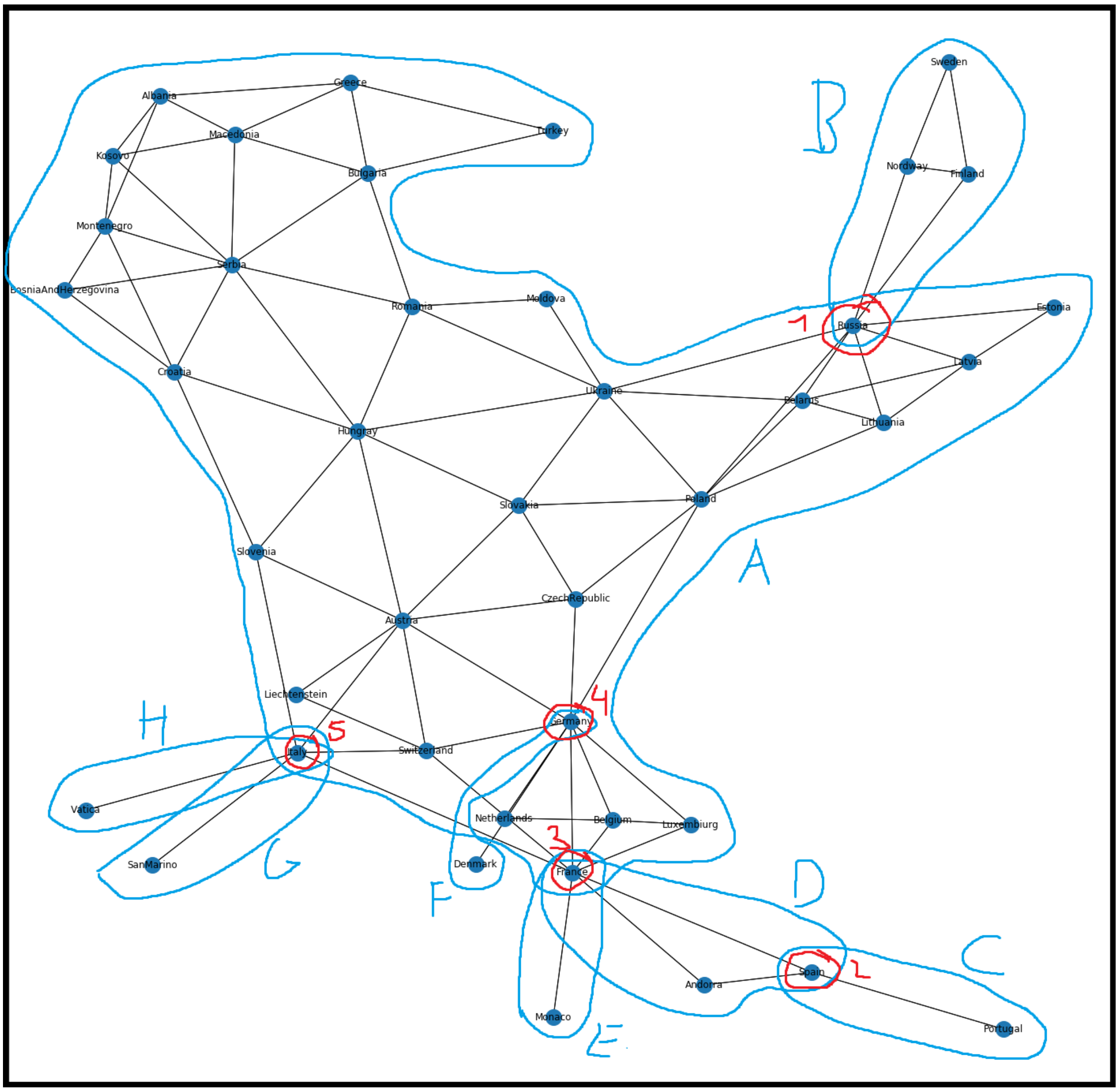
Эйлеров путь в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.

Эйлеров цикл — эйлеров путь, являющийся циклом, то есть замкнутый путь, проходящий через каждое ребро графа ровно по одному разу.

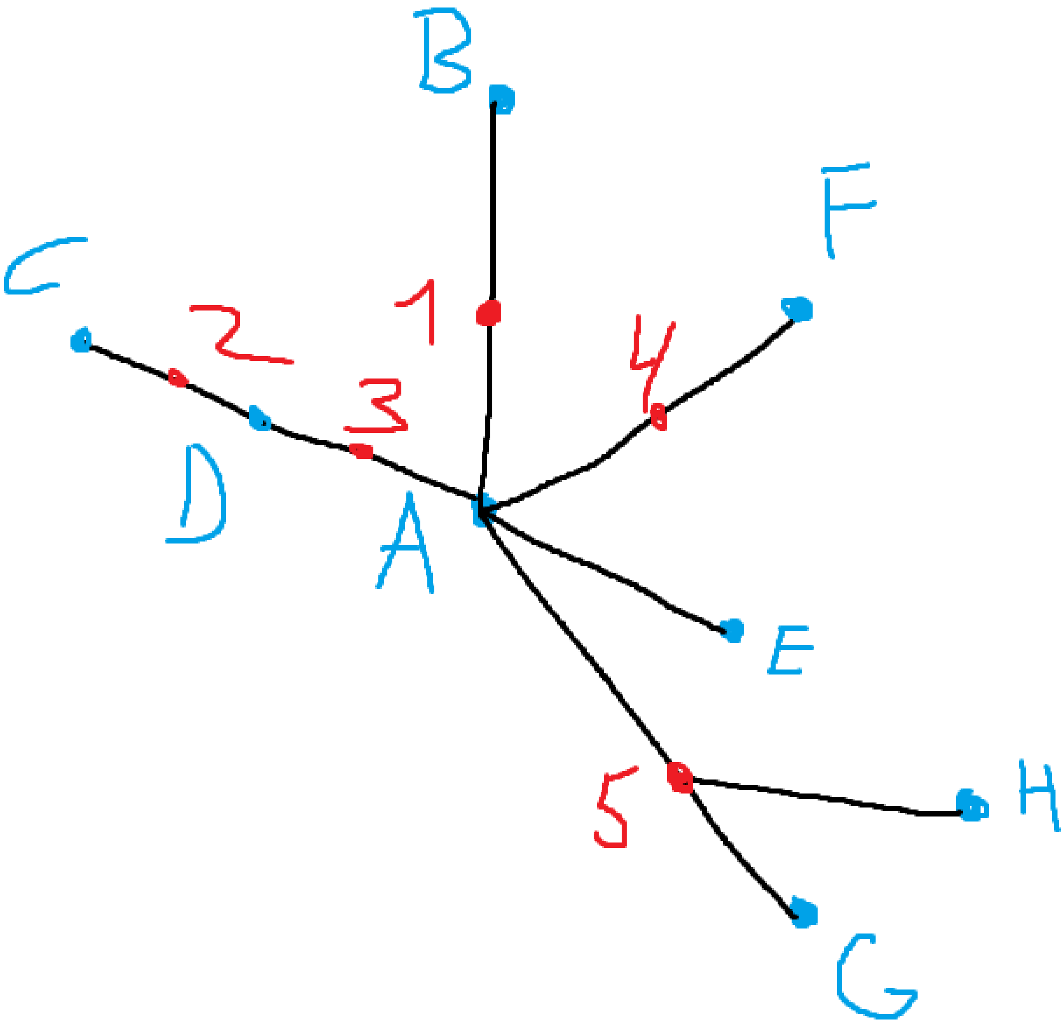
```
a = list(nx.eulerian_circuit(nx.eulerize(G)))
for i in a:
    v1, v2 = i
    print('(', v1, '--->', v2, end='), ')
```

```
( Austria ---> Italy), ( Italy ---> SanMarino), ( SanMarino ---> Italy), ( Italy ---> Vatican), ( Vatican ---> Italy), ( Italy
---> France), ( France ---> Monaco), ( Monaco ---> France), ( France ---> Spain), ( Spain ---> Portugal), ( Portugal ---> Spai
n), ( Spain ---> Andorra), ( Andorra ---> France), ( France ---> Luxembiurg), ( Luxembiurg ---> Belgium), ( Belgium ---> Nether
lands), ( Netherlands ---> Germany), ( Germany ---> Switzerland), ( Switzerland ---> Italy), ( Italy ---> Slovenia), ( Slovenia
---> Croatia), ( Croatia ---> Serbia), ( Serbia ---> Montenegro), ( Montenegro ---> Kosovo), ( Kosovo ---> Serbia), ( Serbia --
-> Romania), ( Romania ---> Moldova), ( Moldova ---> Ukraine), ( Ukraine ---> Russia), ( Russia ---> Finland), ( Finland ---> N
ordway), ( Nordway ---> Finland), ( Finland ---> Sweden), ( Sweden ---> Nordway), ( Nordway ---> Russia), ( Russia ---> Polan
d), ( Poland ---> Ukraine), ( Ukraine ---> Belarus), ( Belarus ---> Poland), ( Poland ---> Lithuania), ( Lithuania ---> Russi
a), ( Russia ---> Latvia), ( Latvia ---> Lithuania), ( Lithuania ---> Belarus), ( Belarus ---> Russia), ( Russia ---> Estonia),
( Estonia ---> Latvia), ( Latvia ---> Belarus), ( Belarus ---> Poland), ( Poland ---> Slovakia), ( Slovakia ---> Ukraine), ( Uk
raine ---> Romania), ( Romania ---> Bulgaria), ( Bulgaria ---> Serbia), ( Serbia ---> BosniaAndHerzegovina), ( BosniaAndHerzego
vina ---> Montenegro), ( Montenegro ---> BosniaAndHerzegovina), ( BosniaAndHerzegovina ---> Croatia), ( Croatia ---> Montenegr
o), ( Montenegro ---> Albania), ( Albania ---> Kosovo), ( Kosovo ---> Macedonia), ( Macedonia ---> Bulgaria), ( Bulgaria ---> T
urkey), ( Turkey ---> Greece), ( Greece ---> Macedonia), ( Macedonia ---> Bulgaria), ( Bulgaria ---> Greece), ( Greece ---> Alb
ania), ( Albania ---> Macedonia), ( Macedonia ---> Serbia), ( Serbia ---> Hungray), ( Hungray ---> Slovenia), ( Slovenia ---> A
ustria), ( Austria ---> Switzerland), ( Switzerland ---> France), ( France ---> Luxembiurg), ( Luxembiurg ---> Germany), ( Germ
any ---> Denmark), ( Denmark ---> Germany), ( Germany ---> France), ( France ---> Belgium), ( Belgium ---> Germany), ( Germany
---> Poland), ( Poland ---> CzechRepublic), ( CzechRepublic ---> Slovakia), ( Slovakia ---> Austria), ( Austria ---> Switzerlan
d), ( Switzerland ---> Liechtenstein), ( Liechtenstein ---> Austria), ( Austria ---> Slovakia), ( Slovakia ---> Hungray), ( Hun
gray ---> Ukraine), ( Ukraine ---> Romania), ( Romania ---> Hungray), ( Hungray ---> Croatia), ( Croatia ---> Hungray), ( Hungr
ay ---> Austria), ( Austria ---> Germany), ( Germany ---> CzechRepublic), ( CzechRepublic ---> Austria),
```

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

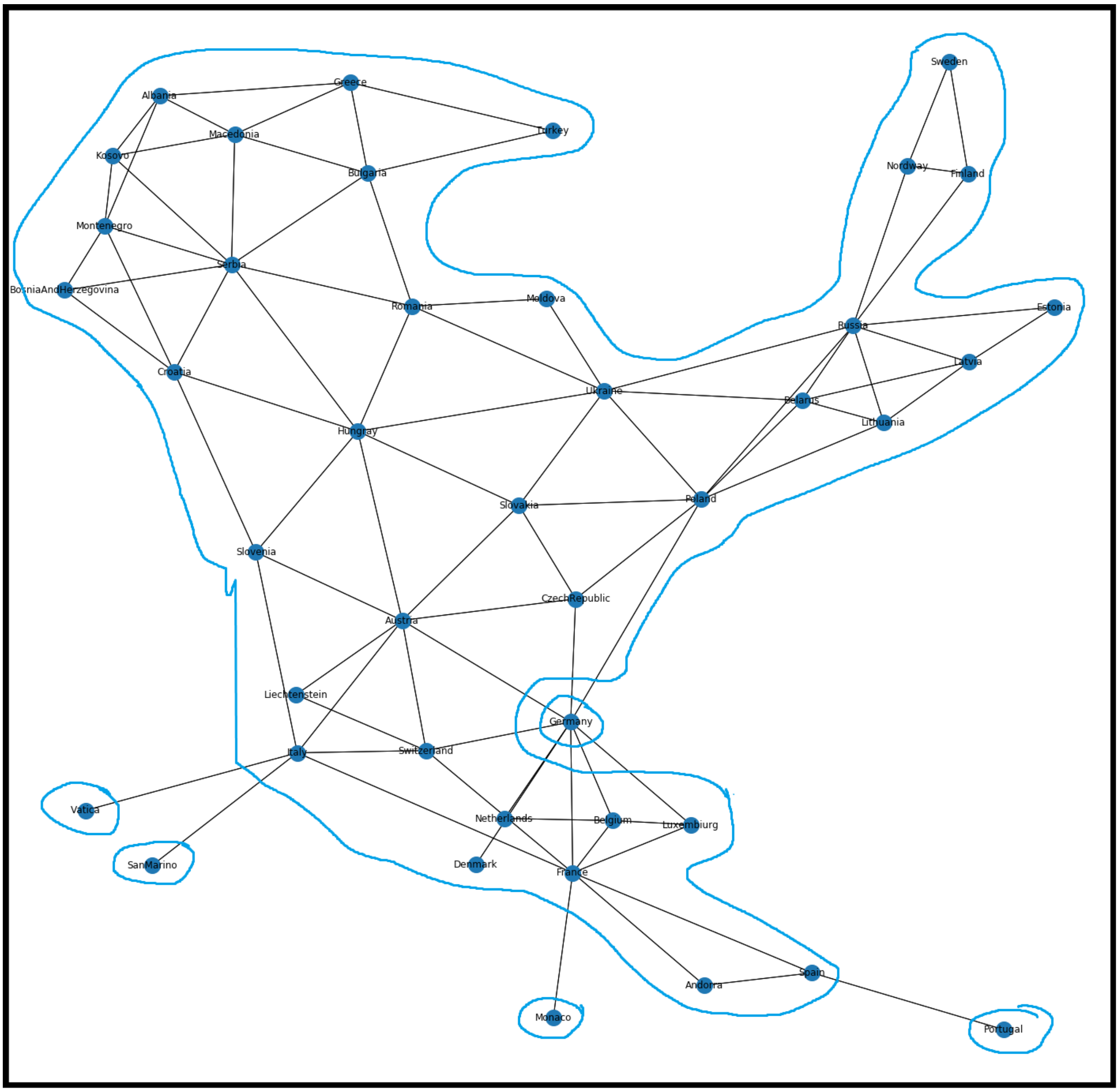


Блоки – синий цвет, шарниры – красный цвет.



B
F
C
1
2
4
3
D
A
E
5
H
G

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

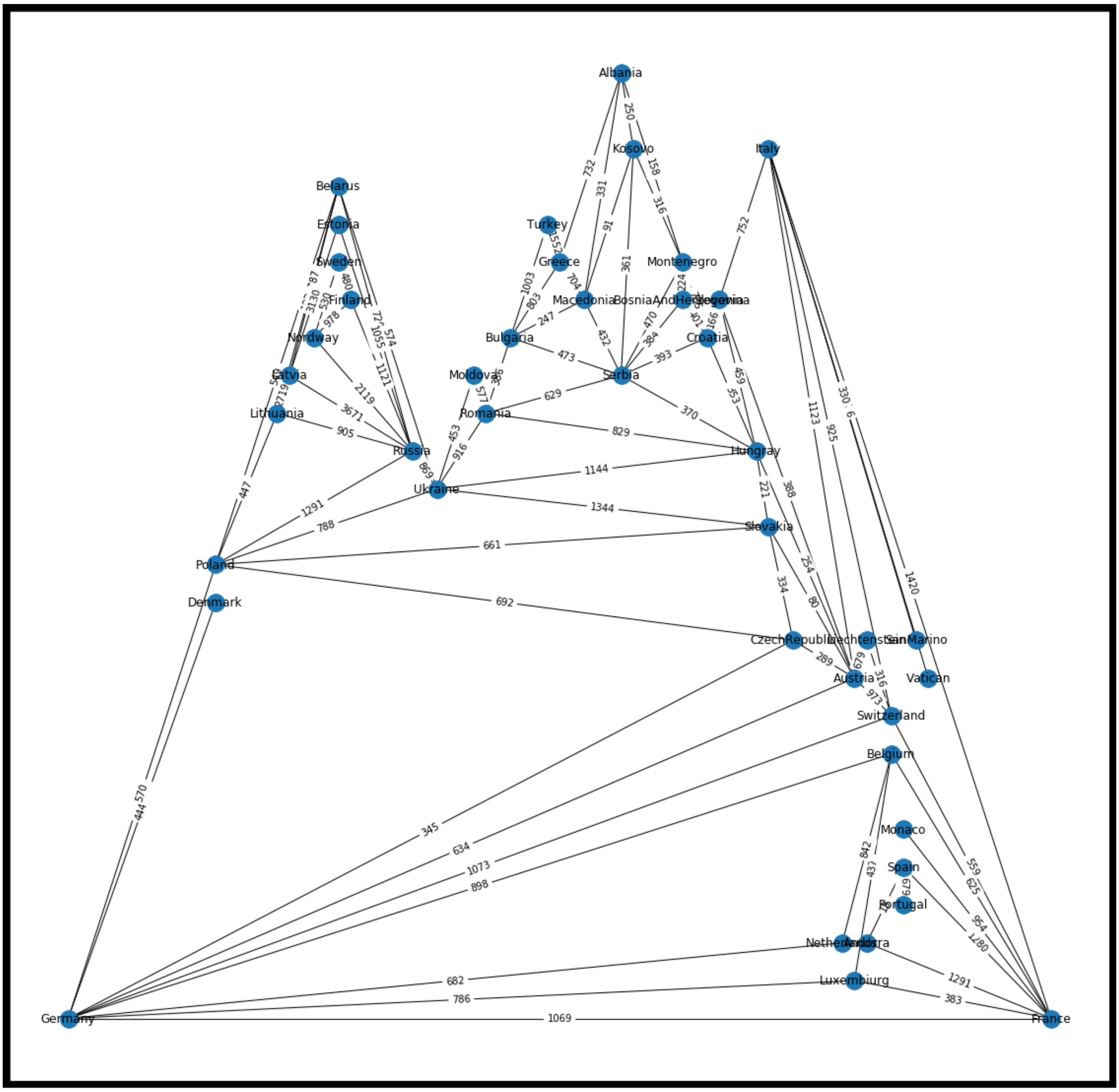


N: Construct an SPQR tree of the largest biconnected component of G.

o: Add the weight function $w : E \rightarrow R$ denoting the distance[3] 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 $G_{*w} = (V, E, w)$.

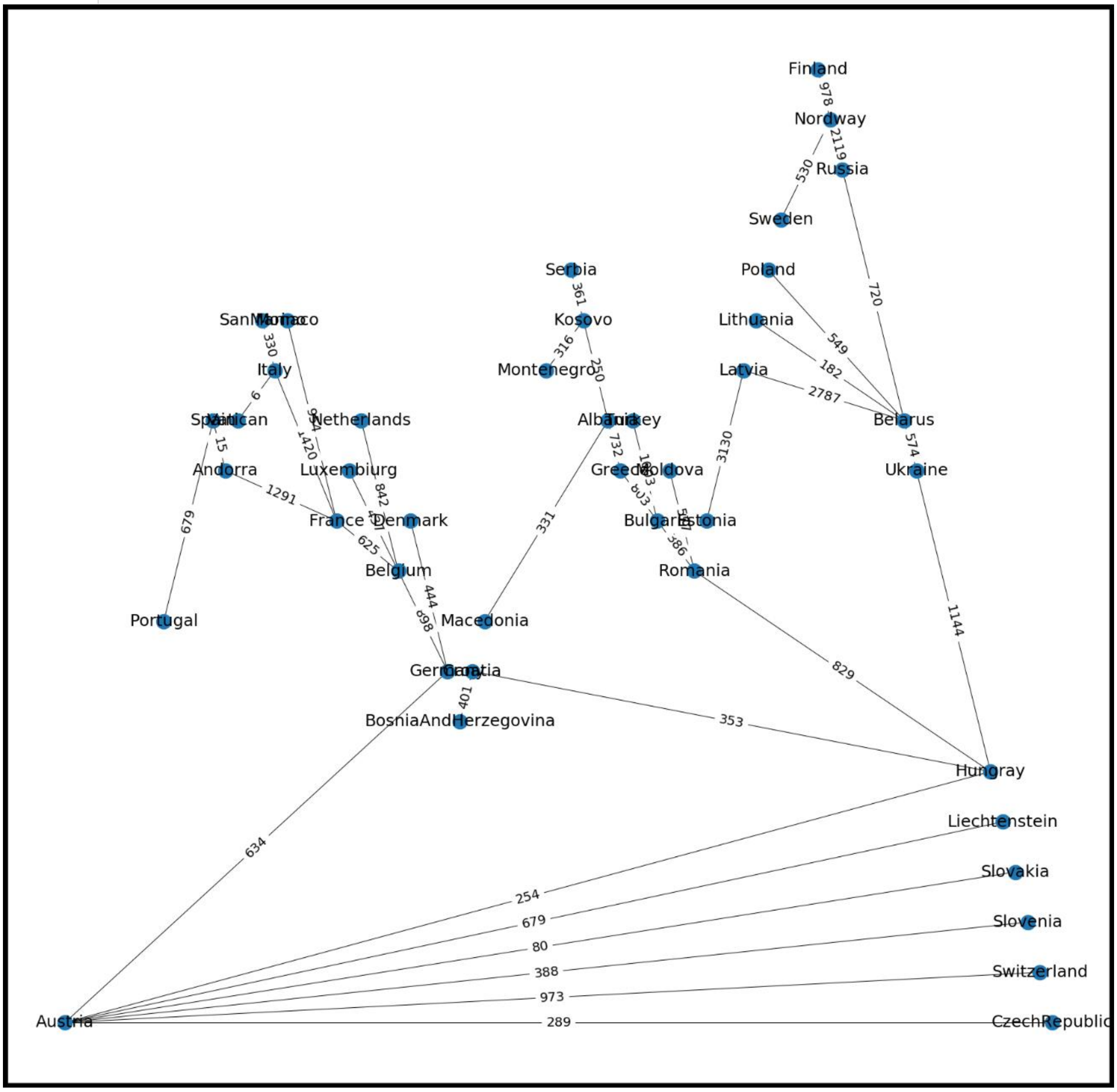
Graph with a weight function:

```
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()
```

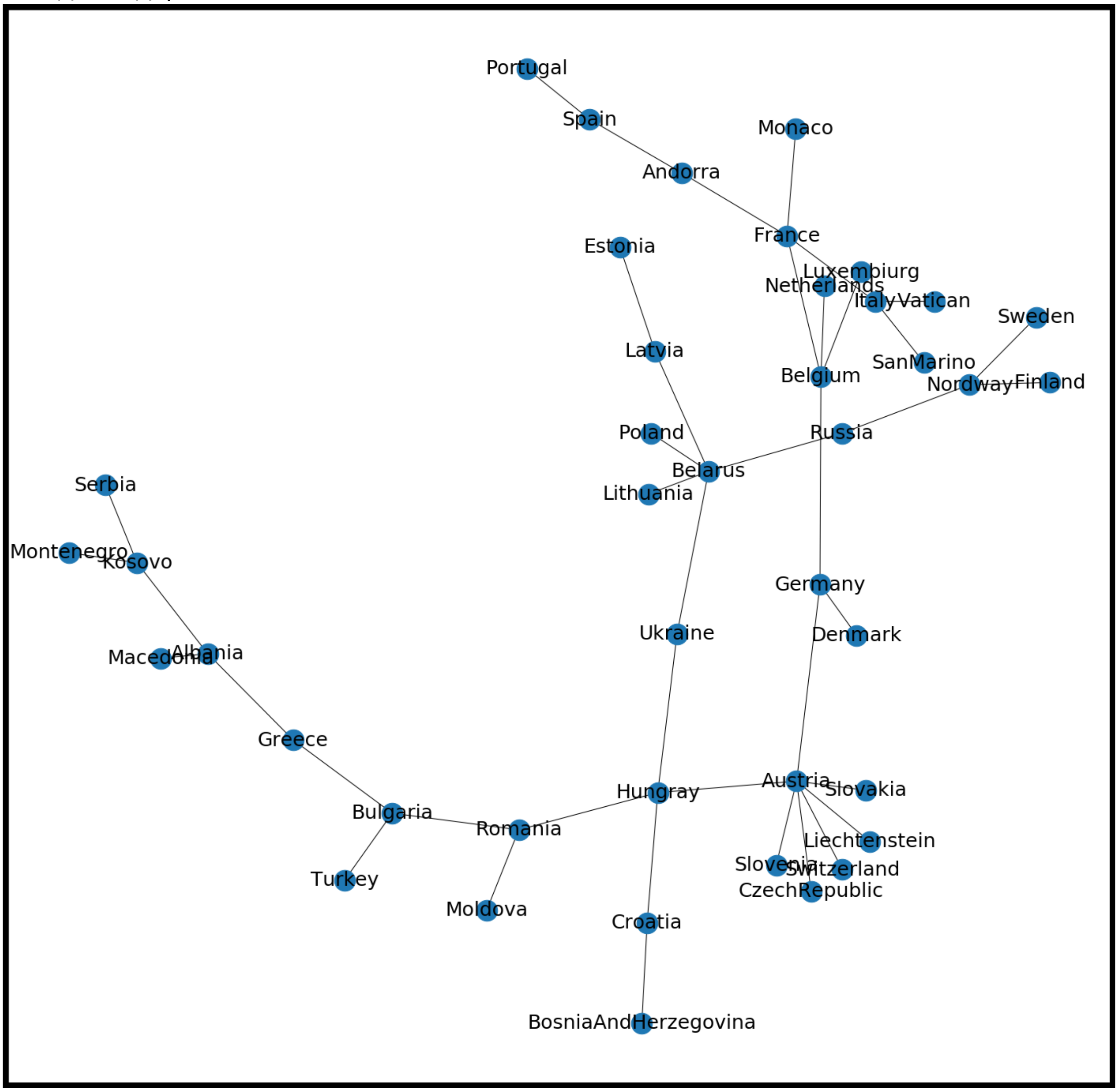


Минимальное остовное дерево, полученное с помощью алгоритма Прима:

In [16]:
```python
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 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 $t1$, $t2$,… , tk, таких что для каждого i: $|ti|$⩽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.

```python
    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, Belg
ium, Belarus, Spain, Italy, Kosovo, Albania, Greece, Bulgaria, Austria, Austria, Andorra, France, Nordway, Russia, Belarus, Ukr
aine, Austria, Bulgaria, Romania, Hungray, Hungray, Austria, Germany, Belgium, France, Italy,
```

Задание №2.

Theorem 1. (Triangle Ineqality) For any connected graph $G = \langle V, E \rangle$:
$\forall x, y, z \in V$: dist(x,y) + dist$(y, z) \geq$ dist(x, z)

Док-во:
Воспользуемся методом “от противного”. Пусть dist(x, y) + dist(y, z) < 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 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 Ineqality for any connected graph справедливо.

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