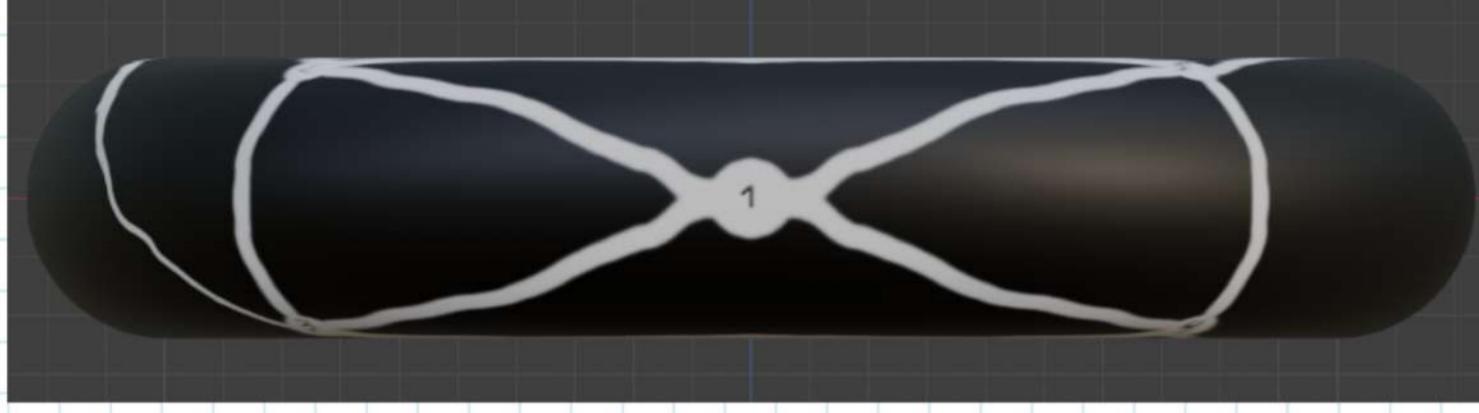


5. Draw K_5 and $K_{3,3}$ on the surface of a torus (a donut-shaped solid) without intersecting edges.

K_5 :

Oy :



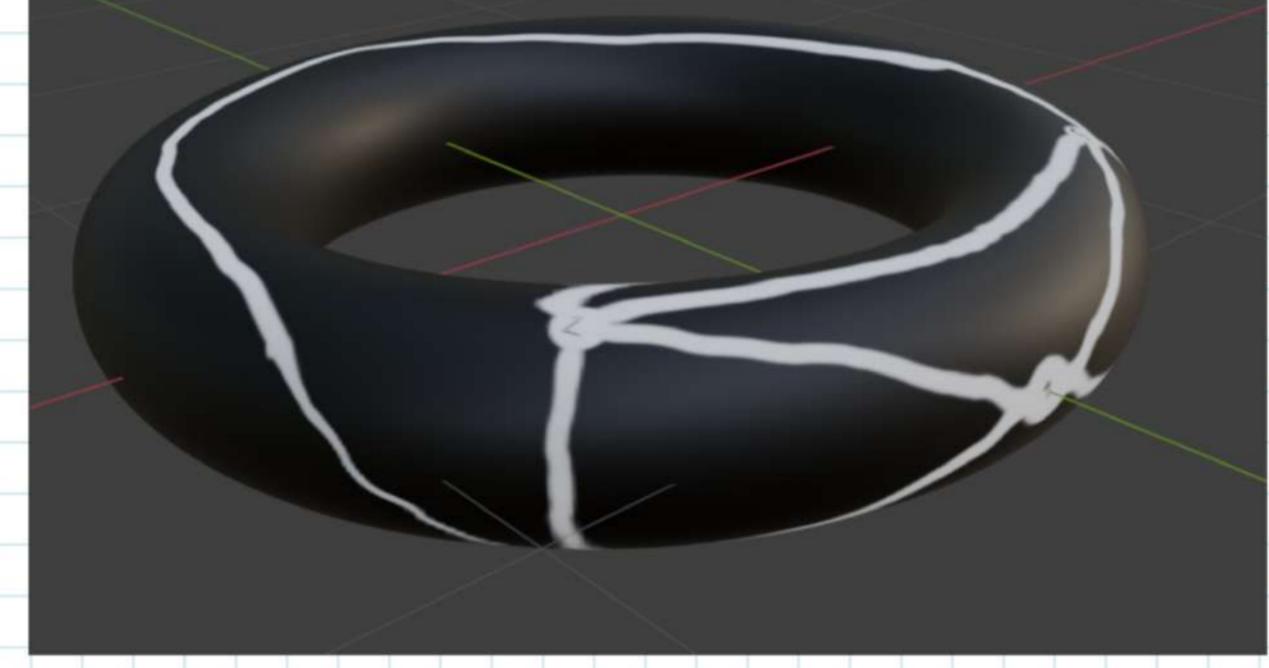
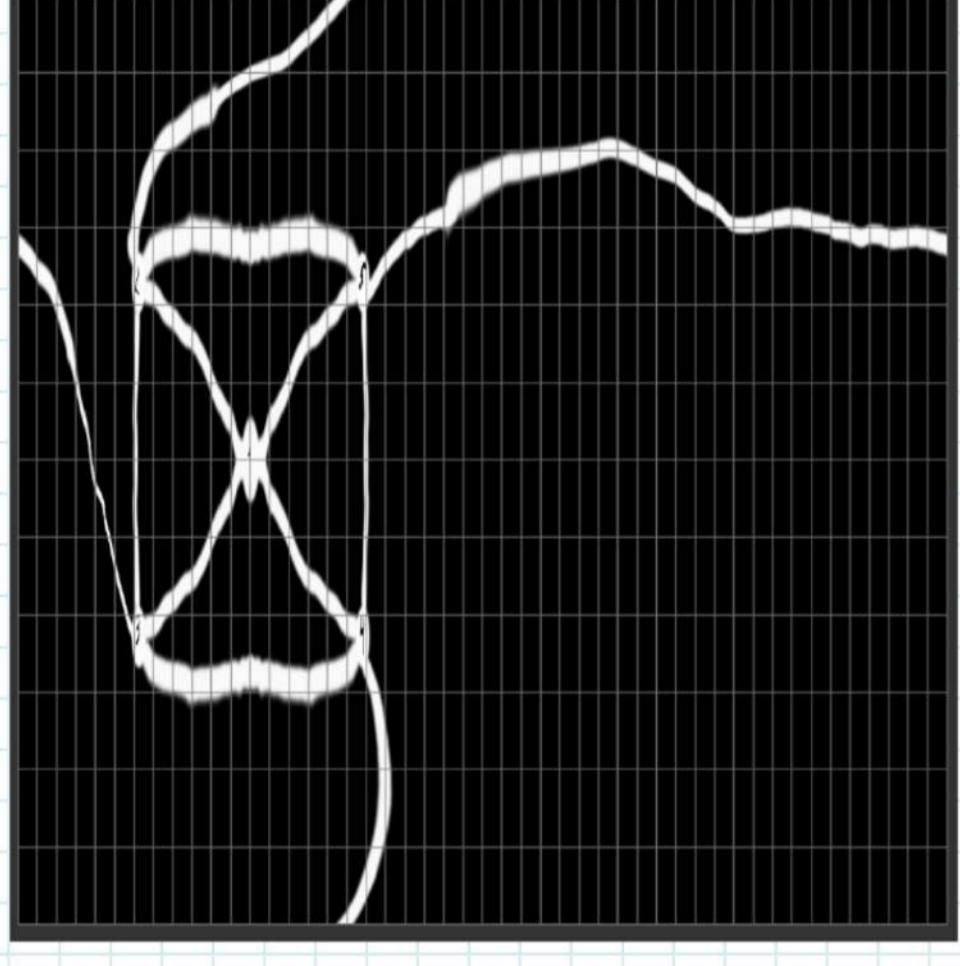
Ox :



Oz :



Parabola:



$K_{3,3}$:

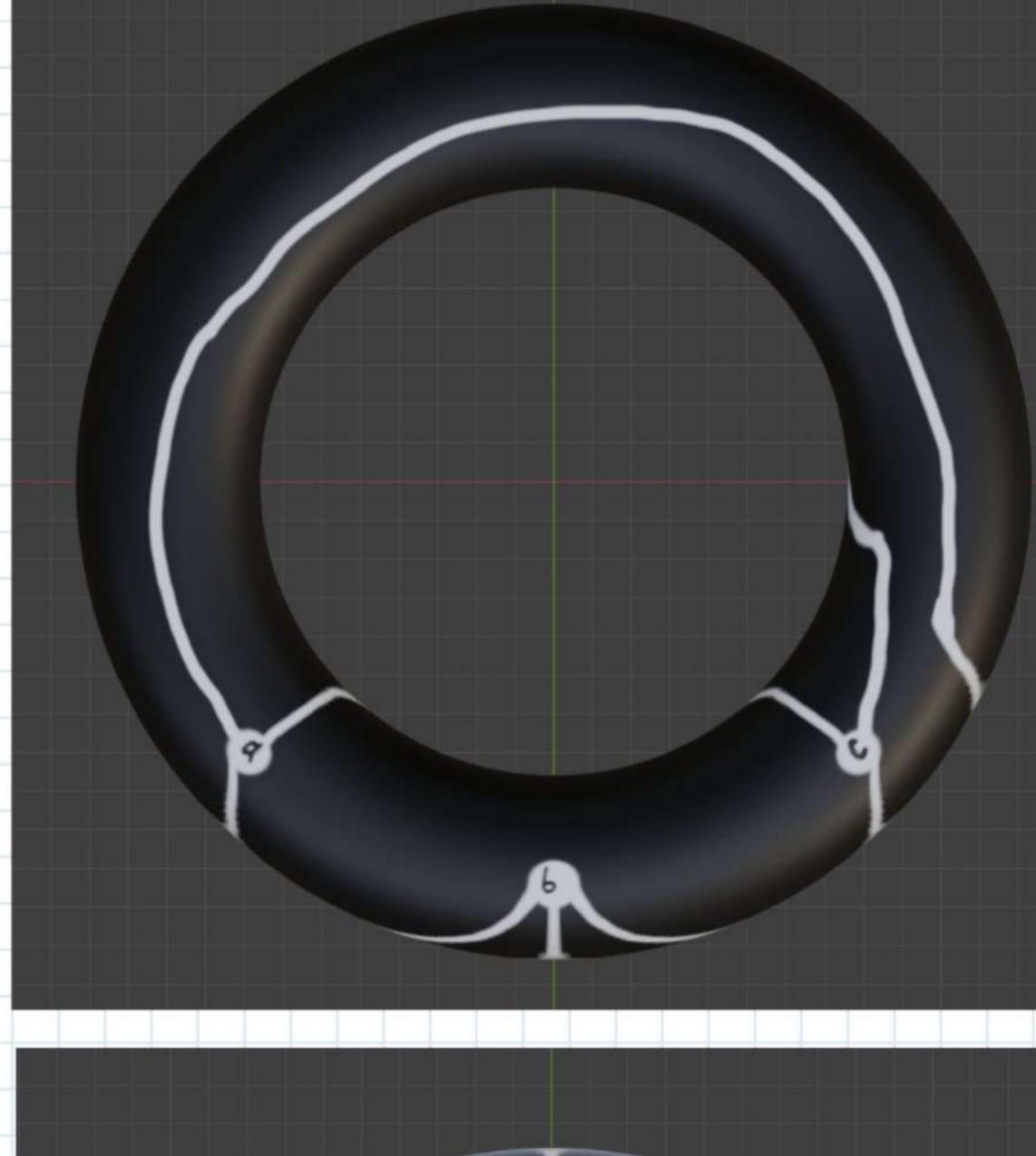
Oy :



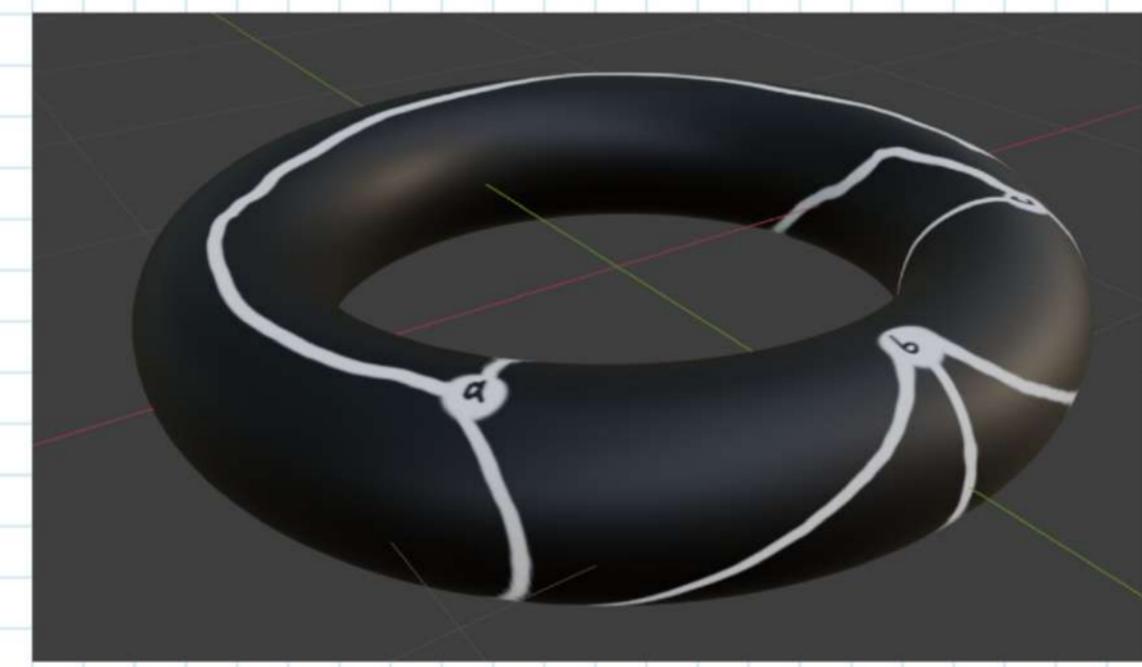
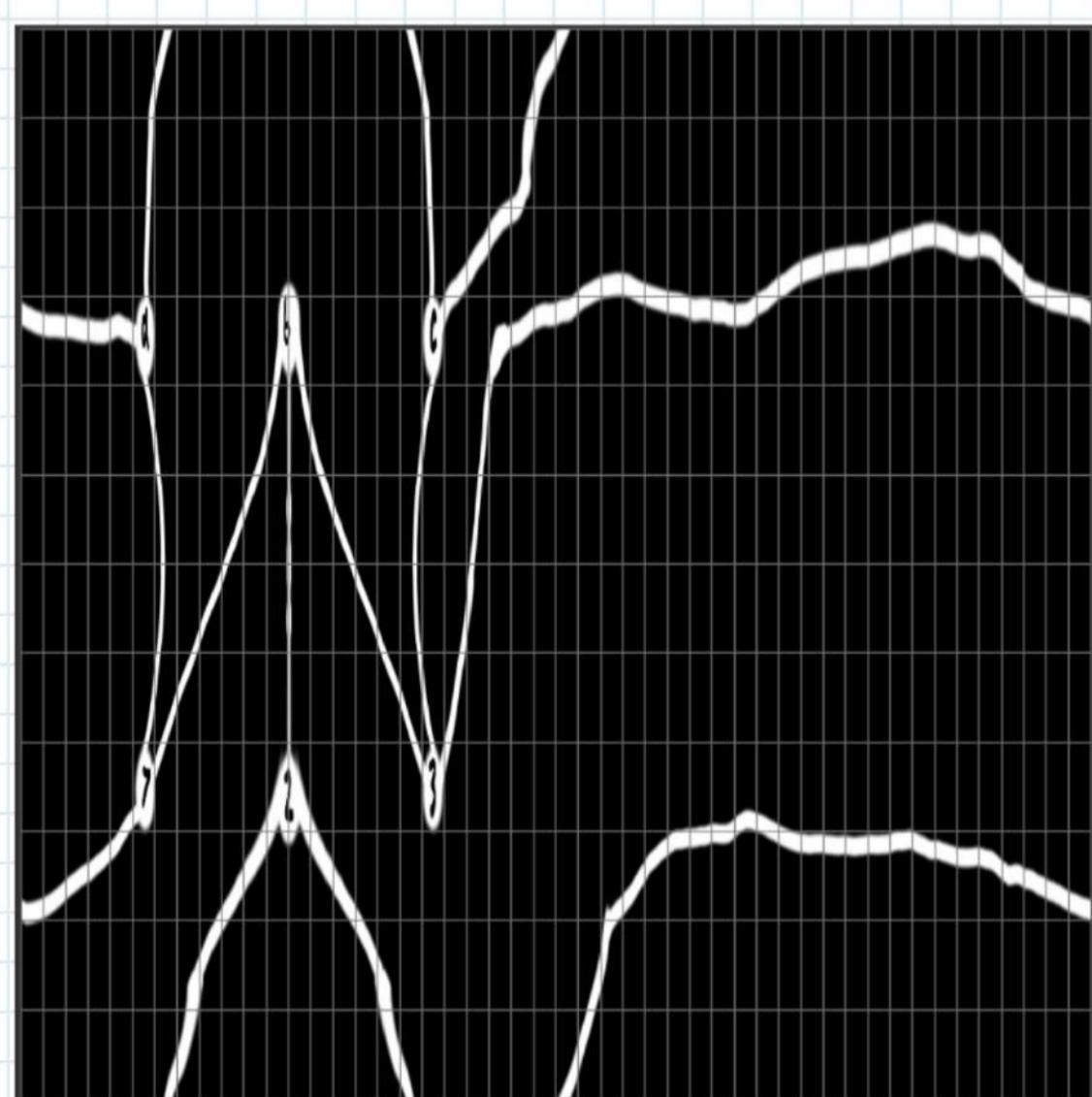
Ox :



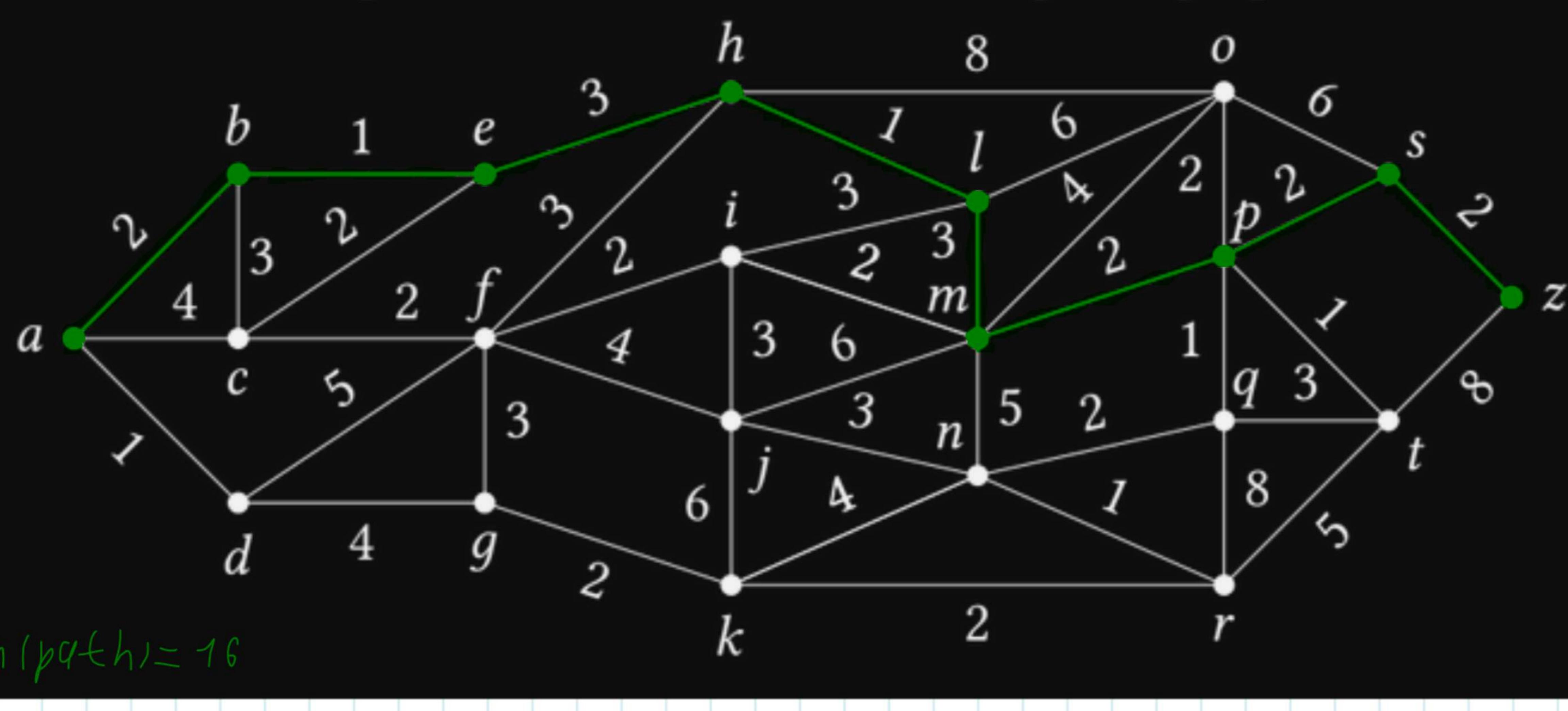
Oz :



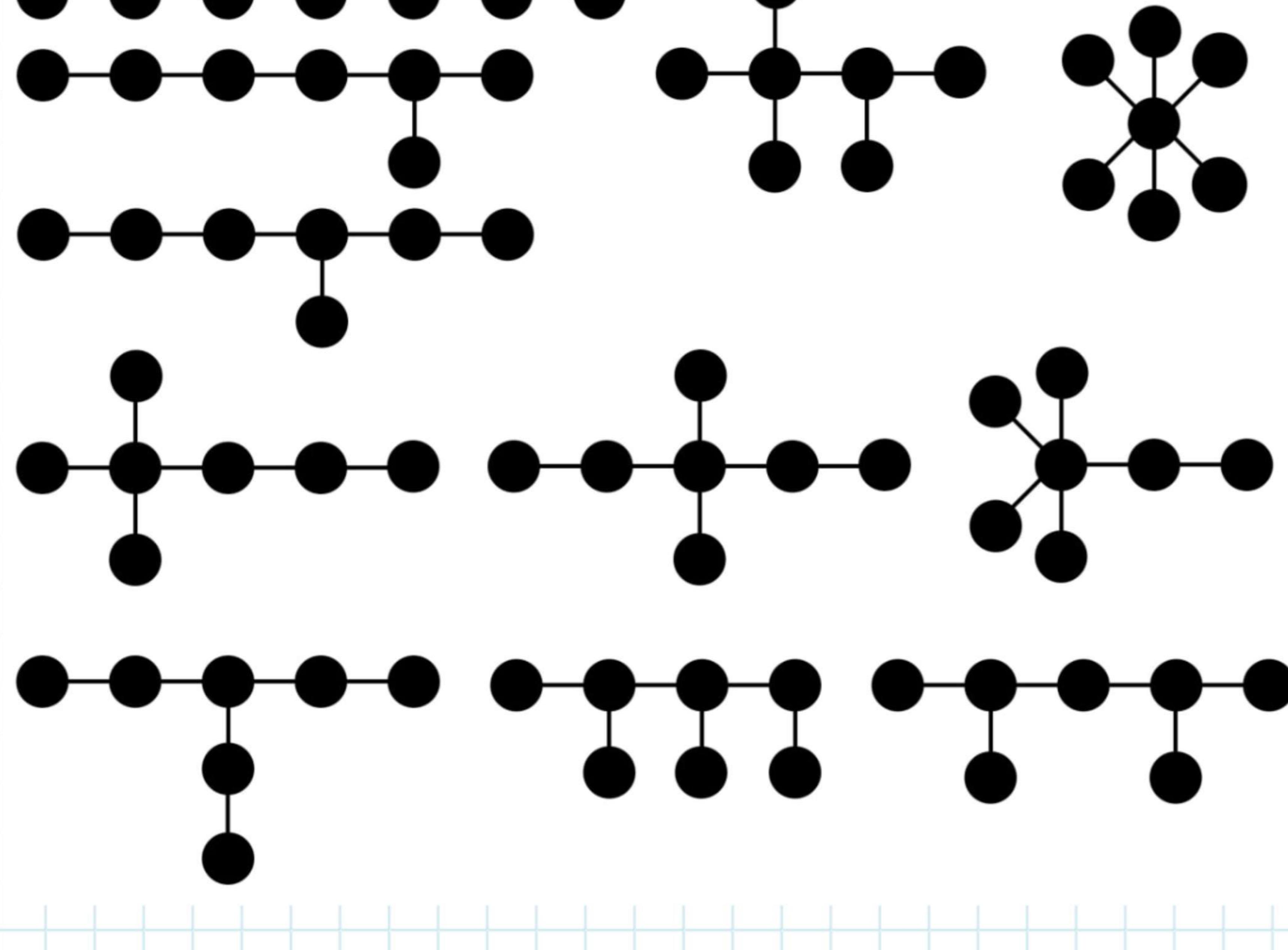
Parabola:



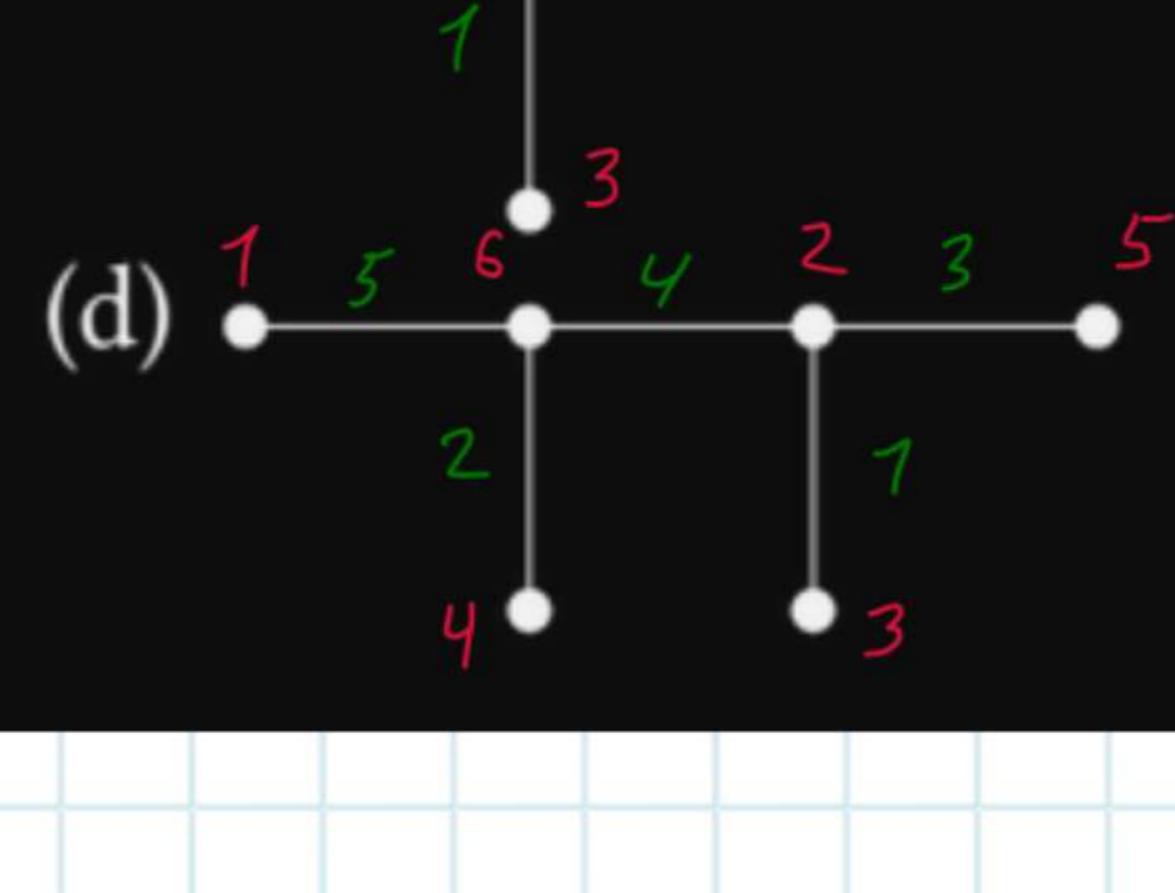
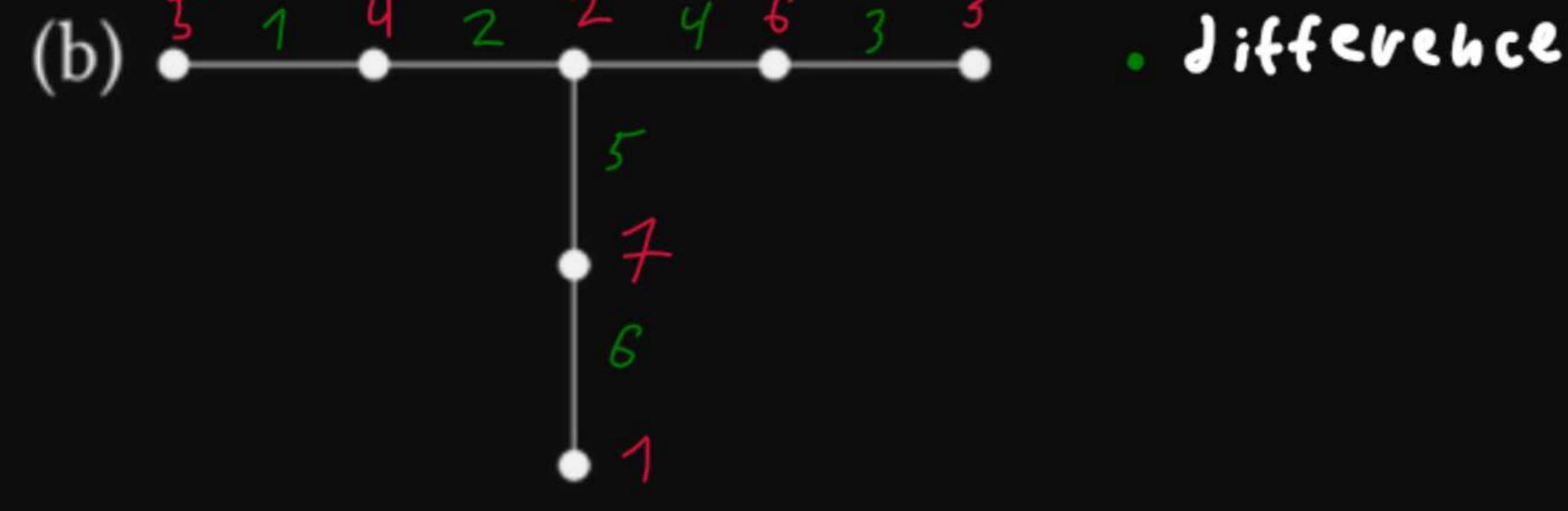
3. Find a shortest path between a and z in the given graph.



9. Draw all pairwise non-isomorphic unlabeled unrooted trees on 7 vertices.



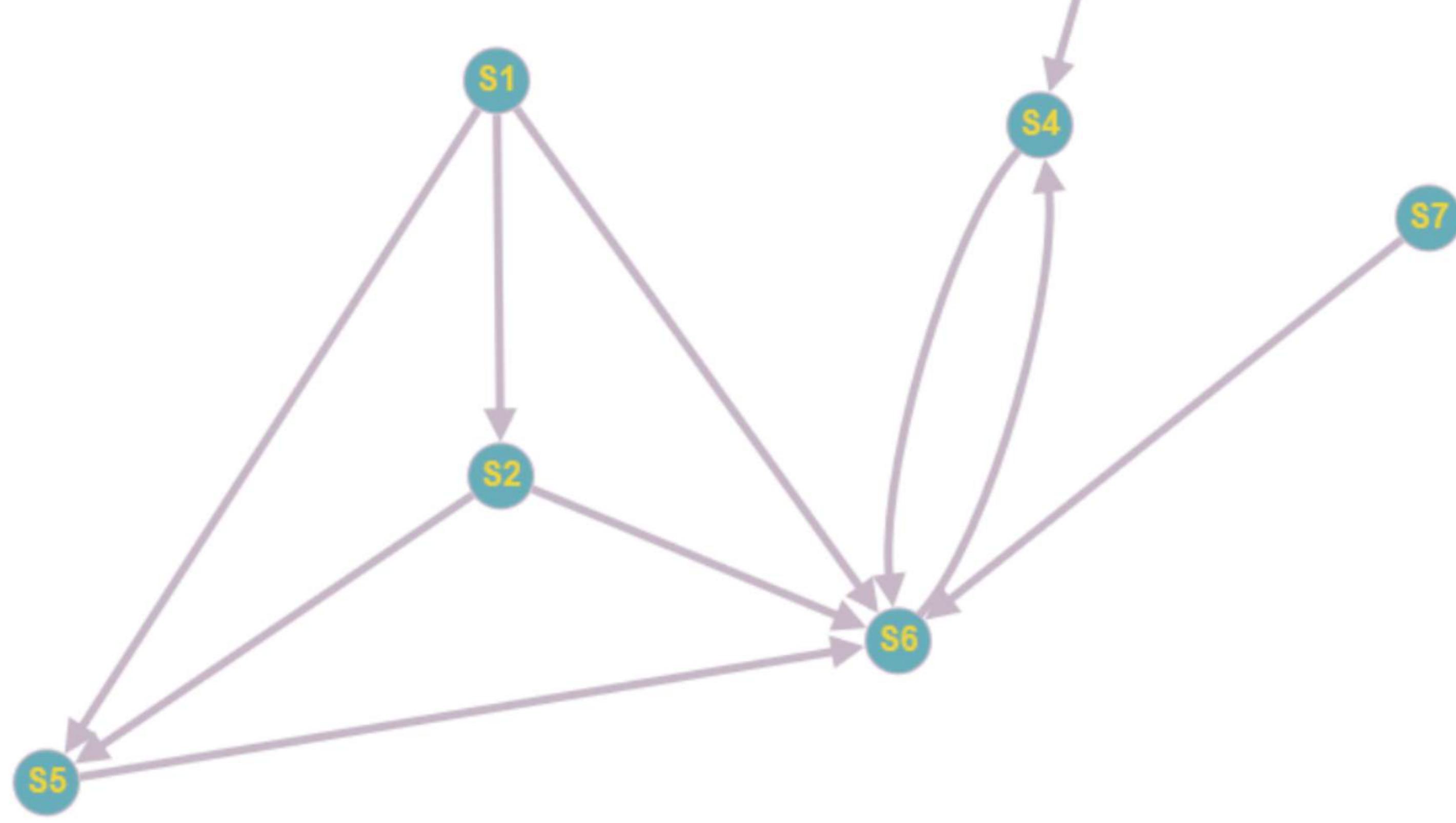
7. A tree with n vertices is called **graceful** if its vertices can be labeled with the integers $1, 2, \dots, n$ in such a way that the absolute values of the difference of the labels of adjacent vertices are all different. Show that the following graphs are graceful.



2. A **precedence graph** is a directed graph where the vertices represent the program instructions and the edges represent the dependencies between instructions: there is an edge from one statement to a second statement if the second statement cannot be executed before the first statement. For example, the instruction $b := a + 1$ depends on the instruction $a := 0$, so there would be an edge from the statement $S_1 = (a := 0)$ to the statement $S_2 = (b := a + 1)$.

Construct a precedence graph for the following program:

$S_1 : x := 0$
 $S_2 : x := x + 1$
 $S_3 : y := 2$
 $S_4 : z := y$
 $S_5 : x := x + 2$
 $S_6 : y := x + z$
 $S_7 : z := 4$



6. Floyd's algorithm (pseudocode given below) can be used to find the shortest path between any two vertices in a weighted connected simple graph.
- Implement Floyd's algorithm in your favorite programming language and use it to find the distance between all pairs of vertices in the weighted graph given in task 3.
 - Prove that Floyd's algorithm determines the shortest distance between all pairs of vertices in a weighted simple graph.
 - Explain in detail (with examples and illustrations) the behavior of the Floyd's algorithm on a graph with negative cycles (a *negative cycle* is a cycle whose edge weights sum to a negative value).
 - Give a big- O estimate of the number of operations (comparisons and additions) used by Floyd's algorithm to determine the shortest distance between every pair of vertices in a weighted simple graph with n vertices.
 - Modify the algorithm to output the actual shortest path between any two given vertices, not just the distance between them.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	z
a	0	2	4	1	3	6	5	6	8	10	7	7	10	10	13	12	12	9	14	13	16
b	2	0	3	3	1	5	7	4	7	9	9	5	8	12	11	10	11	11	12	11	14
c	4	3	0	5	2	2	5	5	4	6	7	6	6	9	10	8	9	9	10	9	12
d	1	3	5	0	4	5	4	7	9	6	8	9	9	13	11	11	8	13	12	15	
e	3	1	2	4	0	4	7	3	6	8	9	4	7	11	10	9	10	11	11	10	13
f	6	5	2	5	4	0	3	3	2	4	5	4	4	7	8	6	7	7	8	7	10
g	5	7	5	4	7	3	0	6	5	7	2	7	7	5	10	8	7	4	10	9	12
h	6	4	5	7	3	3	6	0	4	7	8	1	4	9	7	6	7	10	8	7	10
i	8	7	4	7	6	2	5	4	0	3	7	3	2	6	6	4	5	7	6	5	8
j	10	9	6	9	8	4	7	7	3	0	6	6	5	3	8	6	5	4	8	7	10
k	7	9	7	6	9	5	2	8	7	6	0	9	8	3	8	6	5	2	8	7	10
l	7	5	6	8	4	4	7	1	3	6	9	0	3	8	6	5	6	9	7	6	9
m	10	8	6	9	7	4	7	4	2	5	8	3	0	5	4	2	3	6	4	3	6
n	10	12	9	9	11	7	5	9	6	3	3	8	5	0	5	3	2	1	5	4	7
o	13	11	10	13	10	8	10	7	6	8	8	6	4	5	0	2	3	6	4	3	6
p	12	10	8	11	9	6	8	6	4	6	6	5	2	3	2	0	1	4	2	1	4
q	12	11	9	11	10	7	7	7	6	3	2	3	1	0	3	3	2	5			
r	9	11	9	8	11	7	4	10	7	4	2	9	6	1	6	4	3	0	6	5	8
s	14	12	10	13	11	8	10	8	6	8	8	7	4	5	4	2	3	6	0	3	2
t	13	11	9	12	10	7	9	7	5	7	7	6	3	4	3	1	2	5	3	0	5
z	16	14	12	15	13	10	12	10	8	10	10	9	6	7	6	4	5	8	2	5	0

6) Инициализация: Сначала алгоритм устанавливает расстояния между вершинами как веса ребер, соединяющих их, если ребро существует, и как бесконечность в случае отсутствия ребра. Этот шаг инициализации гарантирует, что алгоритм начинает с правильными начальными расстояниями.

Индуктивное предположение: На каждом шаге алгоритм рассматривает промежуточные вершины и обновляет матрицу расстояний, если найден более короткий путь через промежуточную вершину. Пусть расстояние между вершинами i и j обозначается как $d(i, j)$.

Индуктивное предположение заключается в том, что после k -й итерации алгоритма матрица расстояний содержит кратчайшие пути, учитывая только первые k вершин в качестве промежуточных.

Индуктивный шаг: На каждой итерации алгоритма он проверяет, может ли прохождение через вершину k уменьшить расстояние между вершинами i и j . Если да, то обновляет $d(i, j)$, чтобы отразить этот более короткий путь. Этот шаг гарантирует, что после каждой итерации в матрице расстояний содержатся кратчайшие пути, учитывая первые $k+1$ вершин в качестве промежуточных.

Завершение: После n итераций (где n - количество вершин в графе) алгоритм рассмотрел все вершины как потенциальные промежуточные точки. На этом этапе матрица расстояний содержит кратчайшие пути между всеми парами вершин.

Доказательство корректности: По индукции после каждой итерации алгоритма матрица расстояний содержит кратчайшие пути, учитывая больше промежуточных вершин. Поэтому после финальной итерации матрица расстояний должна содержать кратчайшие пути между всеми парами вершин.

c) Когда в цикле есть оптимиземльный цикл на главной диагонали (то есть расстояние от вершины до её самой) появляются оптимиземльные засечки. Если в цикле будет вершина с оптимиземльными засечками на главной диагонали, то возможен протип по оптимиземльному циклу и уменьшить расстояние.

У нас есть оптимиземльный цикл $i \rightarrow j \rightarrow k \rightarrow i$ где всех вершик в кём будет $-\infty$ расстояние, т.к. мы можем беско ходить по нему и уменьшить расстояние, аналогично было бы со всеми вершинами в которых мы можем бы попасть из цикла.

d)
1 for $i := 1$ to n do
2 for $j := 1$ to n do
3 $d(v_i, v_j) := w(v_i, v_j)$ h^2 writes
4 for $i := 1$ to n do
5 for $j := 1$ to n do
6 for $k := 1$ to n do h^3 compares
7 if $d(v_j, v_i) + d(v_i, v_k) < d(v_j, v_k)$ then
8 $d(v_j, v_k) := d(v_j, v_i) + d(v_i, v_k)$

$\Theta(h^3 + h^2) = \Theta(n^3)$

e)

```
def floyd_marshall(graph):
    dist = graph
    num_vertices = len(graph)
    next_vertex = [[None] * num_vertices for _ in range(num_vertices)]
    dist[0][0] = 0
    for i in range(1, num_vertices):
        for j in range(1, num_vertices):
            if dist[i][j] == float('inf'):
                dist[i][j] = dist[i][0] + dist[0][j]
    for k in range(1, num_vertices):
        for i in range(1, num_vertices):
            for j in range(1, num_vertices):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    next_vertex[i][j] = next_vertex[i][k]
    return dist, next_vertex

def reconstruct(next_vertex, start, end):
    path = [start]
    while start != end:
        start = next_vertex[start][end]
        path.append(start)
    path.append(end)
    return path

graph = [[float('inf')] * num_vertices for _ in range(num_vertices)]
while True:
    a, b, weight = input().split()
    if a == '99':
        break
    if a == 'z':
        a = 20
    if b == 'z':
        b = 20
    else:
        a = ord(a) - 97
        b = ord(b) - 97
    weight = int(weight)
    graph[a][b] = graph[b][a] = weight

shortest_distances, next_vertex = floyd_marshall(graph)

start_vertex, end_vertex = input('Get distance between: ').split()
if start_vertex == 'z':
    start_vertex = 20
else:
    start_vertex = ord(start_vertex) - 97
if end_vertex == 'z':
    end_vertex = 20
else:
    end_vertex = ord(end_vertex) - 97

path = reconstruct(next_vertex, start_vertex, end_vertex)
print(f'Shortest path between {start_vertex} and {end_vertex}: {path}')


def reconstruct_path(next_vertex, start, end):
    path = [start]
    while start != end:
        start = next_vertex[start][end]
        path.append(start)
    path.append(end)
    return path

graph = [[float('inf')] * num_vertices for _ in range(num_vertices)]
while True:
    a, b, weight = input().split()
    if a == '99':
        break
    if a == 'z':
        a = 20
    if b == 'z':
        b = 20
    else:
        a = ord(a) - 97
        b = ord(b) - 97
    weight = int(weight)
    graph[a][b] = graph[b][a] = weight

shortest_distances, next_vertex = floyd_marshall(graph)

start_vertex, end_vertex = input('Get distance between: ').split()
if start_vertex == 'z':
    start_vertex = 20
else:
    start_vertex = ord(start_vertex) - 97
if end_vertex == 'z':
    end_vertex = 20
else:
    end_vertex = ord(end_vertex) - 97

path = reconstruct_path(next_vertex, start_vertex, end_vertex)
print(f'Shortest path between {start_vertex} and {end_vertex}: {path}')


distances_df = pandas.DataFrame(shortest_distances)
distances_df.to_excel('distances.xlsx')
```

text ver:

import pandas

num_vertices = 21

def floyd_marshall(graph):

```
dist = graph
next_vertex = [[None] * num_vertices for _ in range(num_vertices)]
for i in range(num_vertices):
    dist[i][i] = 0
    next_vertex[i][i] = i
for j in range(num_vertices):
    for i in range(num_vertices):
        if graph[i][j] != float('inf'):
            next_vertex[i][j] = i
for k in range(num_vertices):
    for i in range(num_vertices):
        for j in range(num_vertices):
            if dist[i][j] > dist[i][k] + dist[k][j]:
                dist[i][j] = dist[i][k] + dist[k][j]
                next_vertex[i][j] = next_vertex[i][k]
return dist, next_vertex
```

def reconstruct_path(next_vertex, start, end):

```
path = [start]
while start != end:
    start = next_vertex[start][end]
    path.append(start)
return path
```

graph = [[float('inf')] * num_vertices for _ in range(num_vertices)]

```
while True:
    a, b, weight = input().split()
    if a == '99':
        break
    if a == 'z':
        a = 20
    if b == 'z':
        b = 20
    else:
        a = ord(a) - 97
        b = ord(b) - 97
    weight = int(weight)
    graph[a][b] = graph[b][a] = weight
```

shortest_distances, next_vertex = floyd_marshall(graph)

start_vertex, end_vertex = input('Get distance between: ').split()

if start_vertex == 'z':
 start_vertex = 20

else:
 start_vertex = ord(start_vertex) - 97

if end_vertex == 'z':
 end_vertex = 20

else:
 end_vertex = ord(end_vertex) - 97

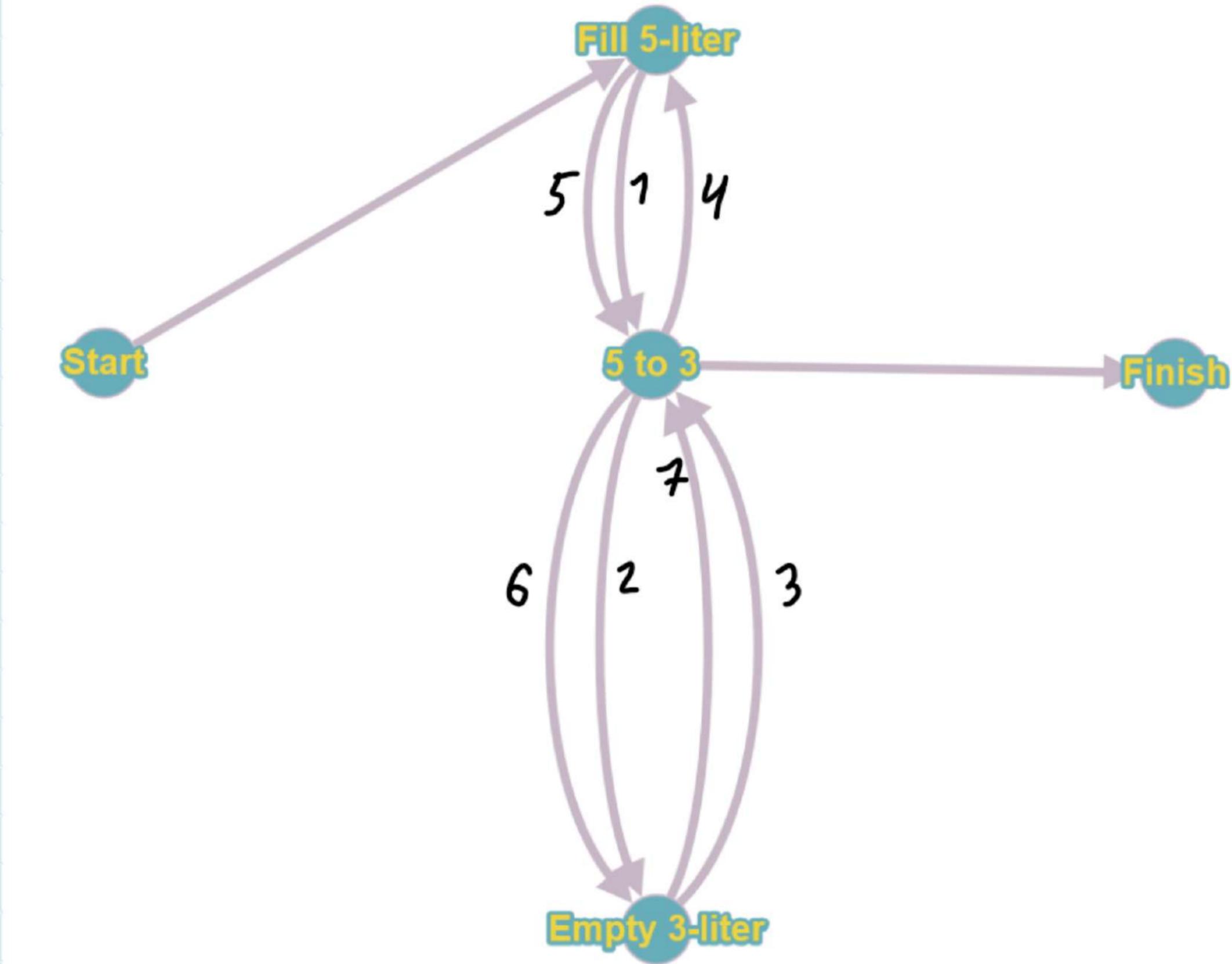
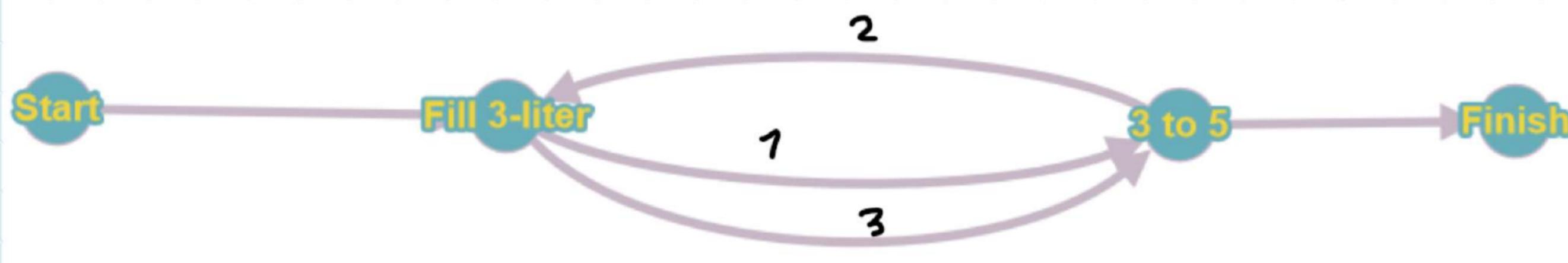
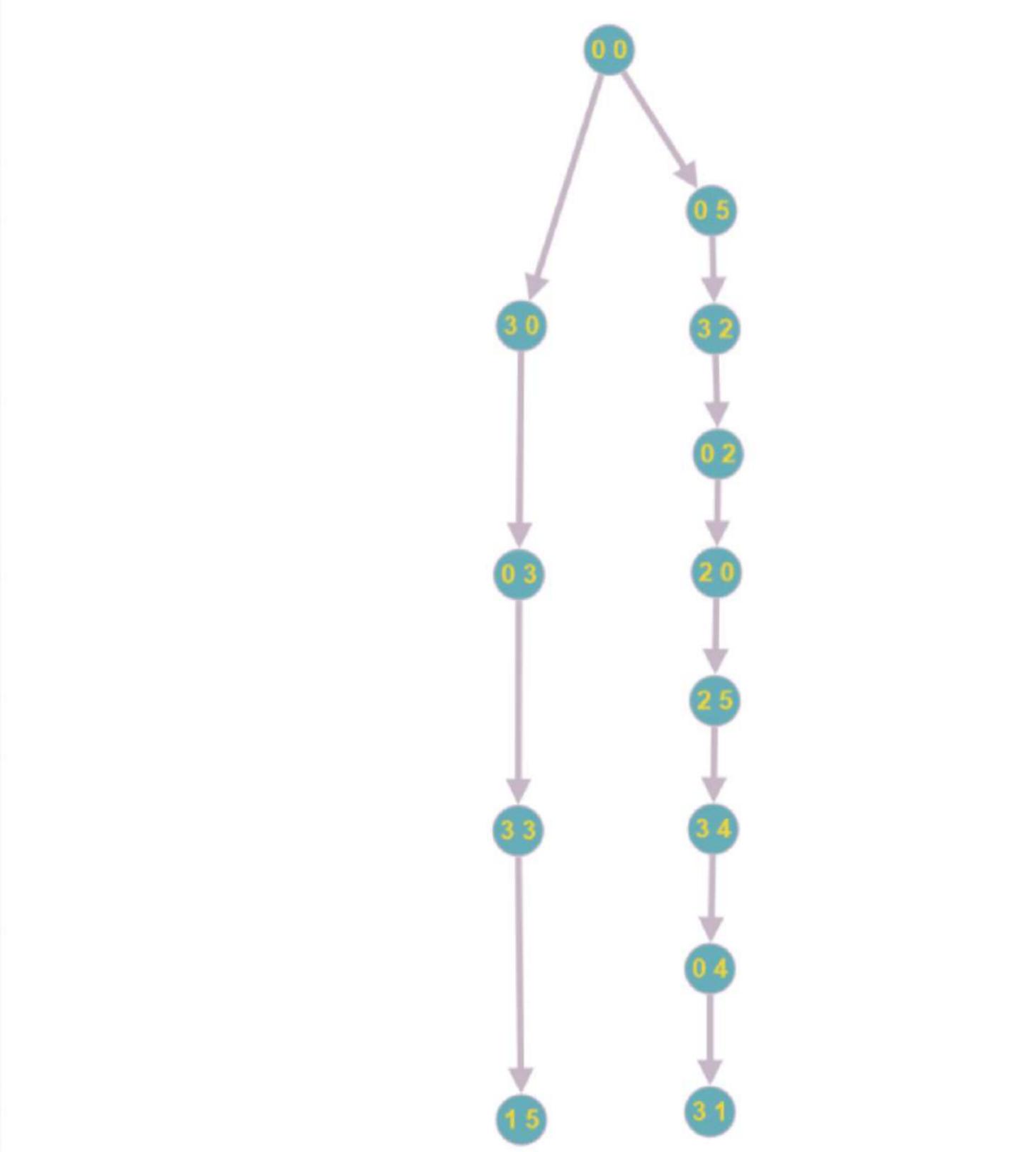
path = reconstruct_path(next_vertex, start_vertex, end_vertex)

print(f'Shortest path between {start_vertex} and {end_vertex}: {path}')

distances_df = pandas.DataFrame(shortest_distances)
distances_df.to_excel('distances.xlsx')

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	z
0	3	4	5	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	0	1	2	5	4	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-
4	1	0	3	4	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	2	3	0	3	2	-	-	-</												

4. Imagine that you have a three-liter jar and another five-liter jar. You can fill any jar with water, empty any jar, or transfer water from one jar to the other. Use a directed graph to demonstrate how you can end up with a jar containing exactly one litre of water.



8. A **caterpillar** is a tree that contains a simple path such that every vertex not contained in this path is adjacent to a vertex in the path.

- (a) Which of the graphs in task 7 are caterpillars?
 - (b) How many non-isomorphic caterpillars are there with six vertices?
 - (c) Prove or disprove that all caterpillars are graceful.

a) d, c, d

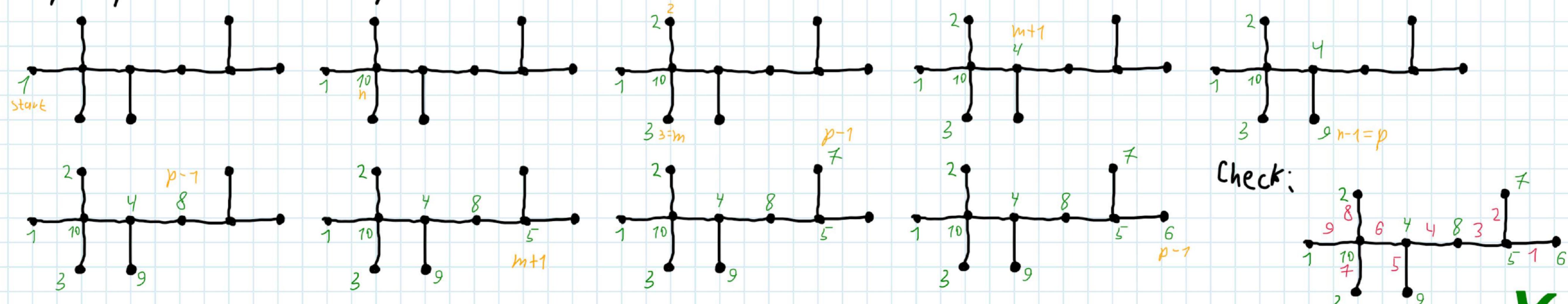
b) 6

С) Ізоморфність еквівалентних caterpillar'ів з n вершинами

Startime с первою vertex, назовем её как 1, а следующую с нею vertex как h , где номер vertex'и h как $2, 3, \dots, m$, следующую vertex в melt как $m+1$, а её номер $h-1, h-2, \dots, p$, следующую vertex в melt как $p-1$. И будем повторять этот алгоритм, пока не получим все vertex. При такой последовательности различия будут

h-1, h-2 ... 1 .

- comments about added labels
 - vertex label
 - difference



11. The **density** of an undirected graph G is the number of edges of G divided by the number of possible edges in an undirected graph with $|G|$ vertices. That is, the density of $G = \langle V, E \rangle$ is $\frac{2|E|}{|V|(|V|-1)}$. A family of graphs $G_n, n = 1, 2, \dots$ is **sparse** if the limit of the density of G_n is zero as n grows without bound, while it is **dense** if this proportion approaches a positive real number. For each of these families of graphs, determine whether it is sparse, dense, or neither.

- (a) K_n (complete graph) (c) $K_{n,n}$ (complete bipartite) (e) Q_n (hypercube graph)
 (b) C_n (cycle graph) (d) $K_{3,n}$ (complete bipartite) (f) W_n (wheel graph)

d) **dense**

$$\lim_{n \rightarrow \infty} \frac{\frac{1}{2} \frac{n(n-1)}{2}}{n(n-1)} = 1$$

c) **dense**

$$\lim_{n \rightarrow \infty} \frac{\frac{2n^2}{2}}{n(n-1)} = 2$$

e) **sparse**

$$\lim_{n \rightarrow \infty} \frac{\frac{2^{(n-1)} \cdot n}{2^n} \cdot n}{2^n \cdot (2^n - 1)} = 0$$

b) **sparse**

$$\lim_{n \rightarrow \infty} \frac{2n}{n(n-1)} = 0$$

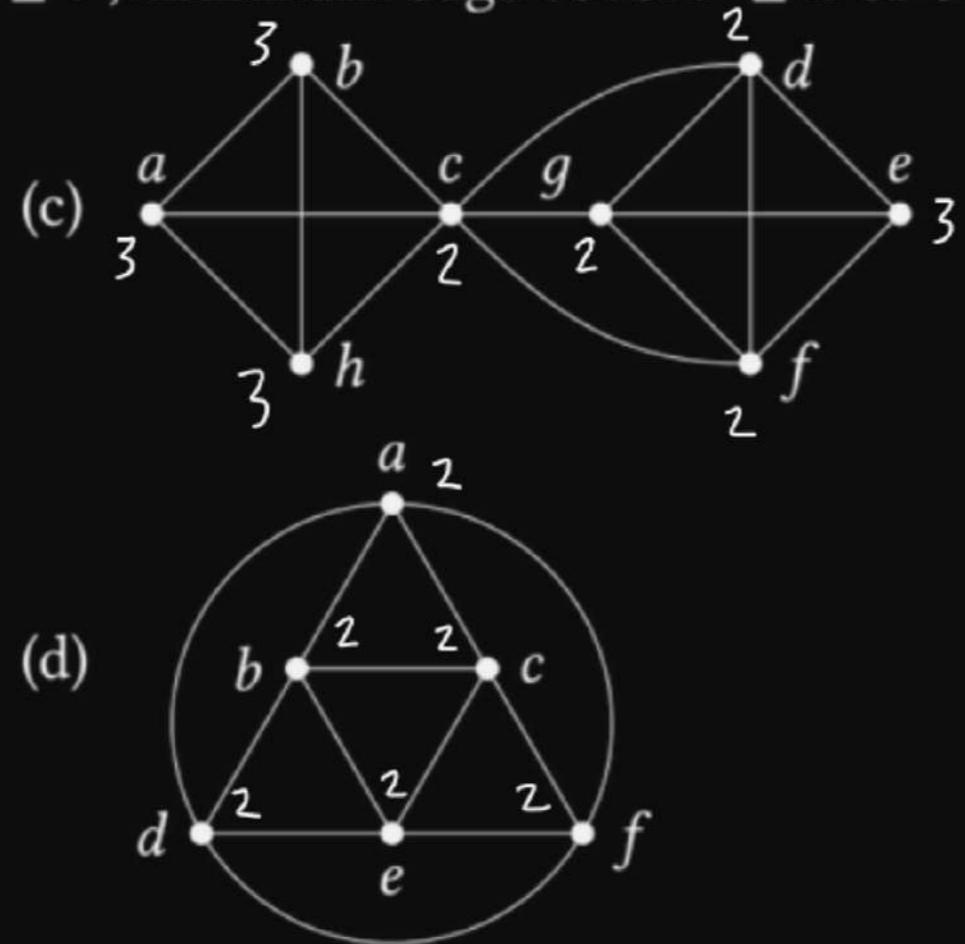
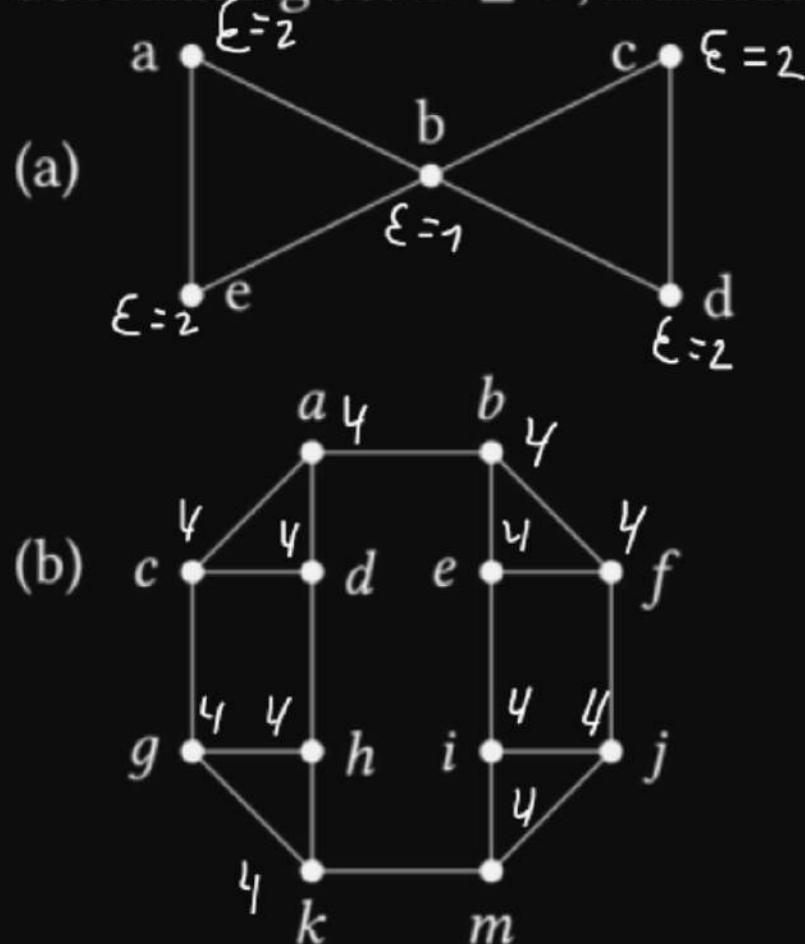
d) **sparse**

$$\lim_{n \rightarrow \infty} \frac{2 \cdot 3n}{n+3(n+2)} = 0$$

f) **sparse**

$$\lim_{n \rightarrow \infty} \frac{2 \cdot 2(n-1)}{n(n-1)} = 0$$

1. For each of the following graphs, find $\alpha(G), \lambda(G), \delta(G), \varepsilon(v)$ of each vertex $v \in V(G)$, $\text{rad}(G), \text{diam}(G), \text{center}(G)$. Find Euler path, Euler circuit, and Hamiltonian cycle, if they exist. In addition, find maximum clique $Q \subseteq V$, maximum stable set $S \subseteq V$, maximum matching $M \subseteq E$, minimum dominating set $D \subseteq V$, minimum vertex cover $R \subseteq V$, minimum edge cover $F \subseteq E$ of G .



d) $\chi - 1$

$\lambda - 2$

$\delta - 2$

$\text{rad} - 1$

$\text{diam} - 2$

center - b

maximum clique - a, b, e

maximum stable set - d, j

maximum matching - ae, cd

minimum dominating set - b

minimum vertex cover - a, b, d

minimum edge cover - ae, bd, cd

Euler path - a, b, c, d, b, e

Euler circuit - a, b, c, d, b, e, a

b) $\chi - 2$

$\lambda - 2$

$\delta - 3$

$\text{rad} - 4$

$\text{diam} - 4$

center - a, b, c, d, e, f, g, h, k, m, j

maximum clique - a, c, d

maximum stable set - a, g, m, e

maximum matching - ab, cd, ef, gh, ij, km

minimum dominating set - d, e, h, i

minimum vertex cover - a, k, d, g, e, j

minimum edge cover - cg, dh, ei, fj, ab, km

Hamiltonian cycle - a, b, e, f, j, i, m, k, g, h, d, c

c) $\chi - 1$

$\lambda - 3$

$\delta - 3$

$\text{rad} - 2$

$\text{diam} - 3$

center - c, d, f, g

maximum clique - a, b, c, h

maximum stable set - d, g

maximum matching - ab, ch, gd, ef

minimum dominating set - c, e

minimum vertex cover - b, d, c, e, a, g

minimum edge cover - ac, bh, df, ge

d) $\chi - 4$

$\lambda - 4$

$\delta - 4$

$\text{rad} - 2$

$\text{diam} - 2$

center - d, b, c, d, e, f

maximum clique - a, d, f

maximum stable set - d, e

maximum matching - df, bc, de

minimum dominating set - a, e

minimum vertex cover - a, b, e, f

minimum edge cover - ab, ce, df

Euler path - a, b, c, a, d, b, f, c, e, d, f, e

Euler circuit - a, b, c, a, d, b, f, c, e, d, f, e, d

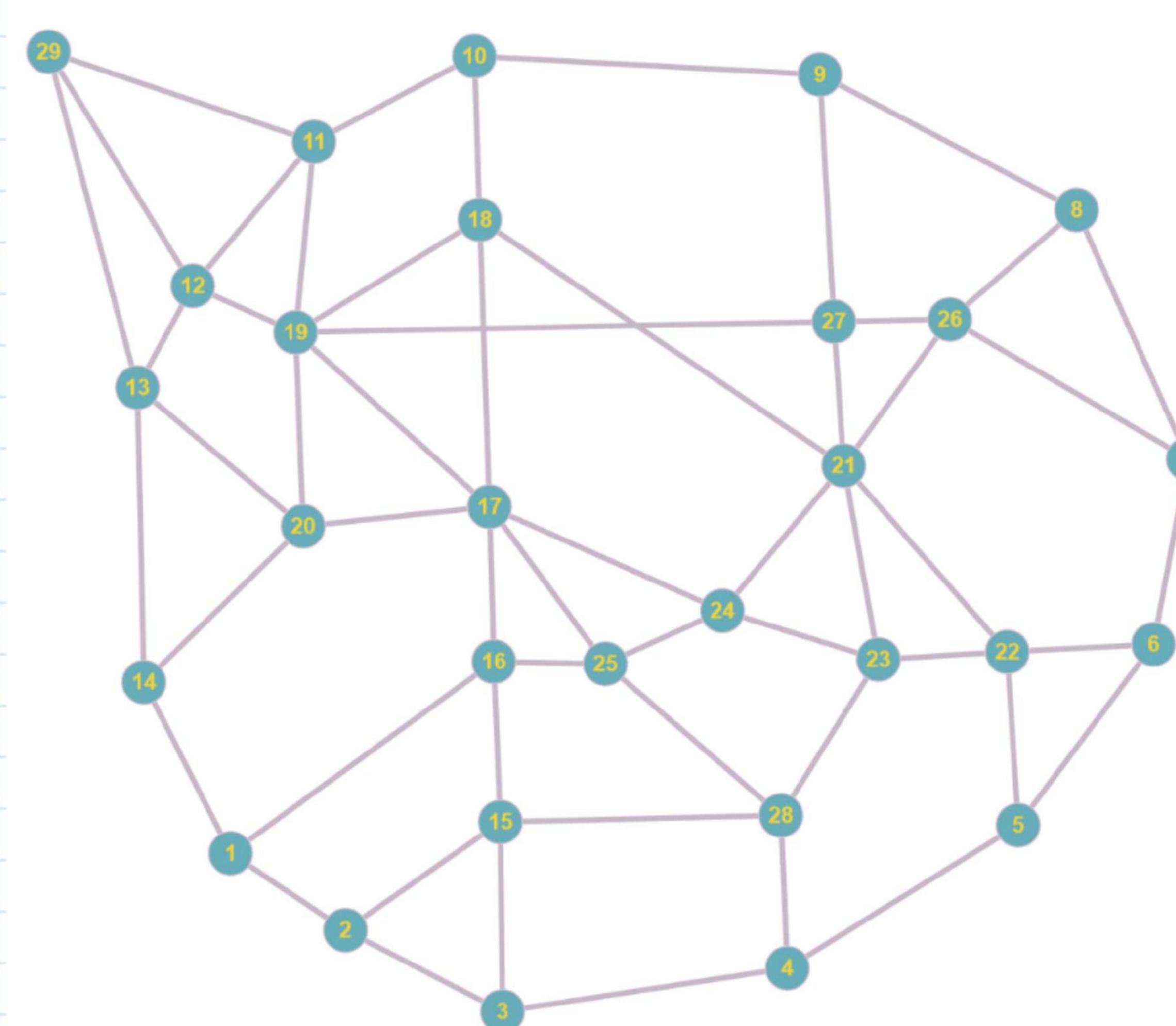
Hamiltonian cycle - a, b, c, e, f, d

12. Consider a graph of subway lines in Saint Petersburg in 2050². Represent each transfer station as a single vertex. Smoothen out³ all 2-degree vertices and remove all pendant (1-degree) vertices (repeat until fixed point). In the resulting graph¹, find vertex and edge connectivity, maximum stable set, maximum matching, minimum dominating set, minimum vertex and edge covers.

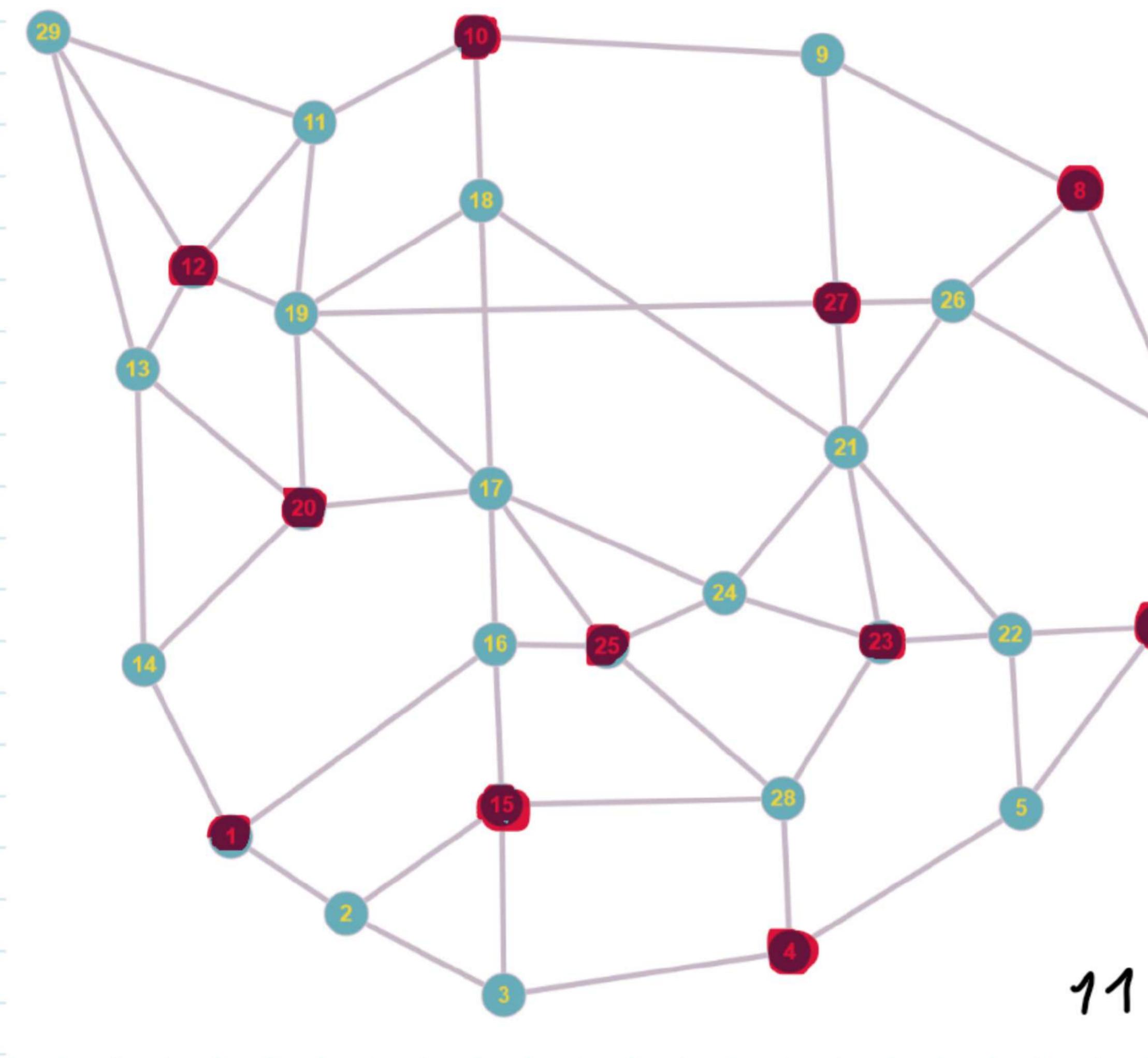
Start graph:



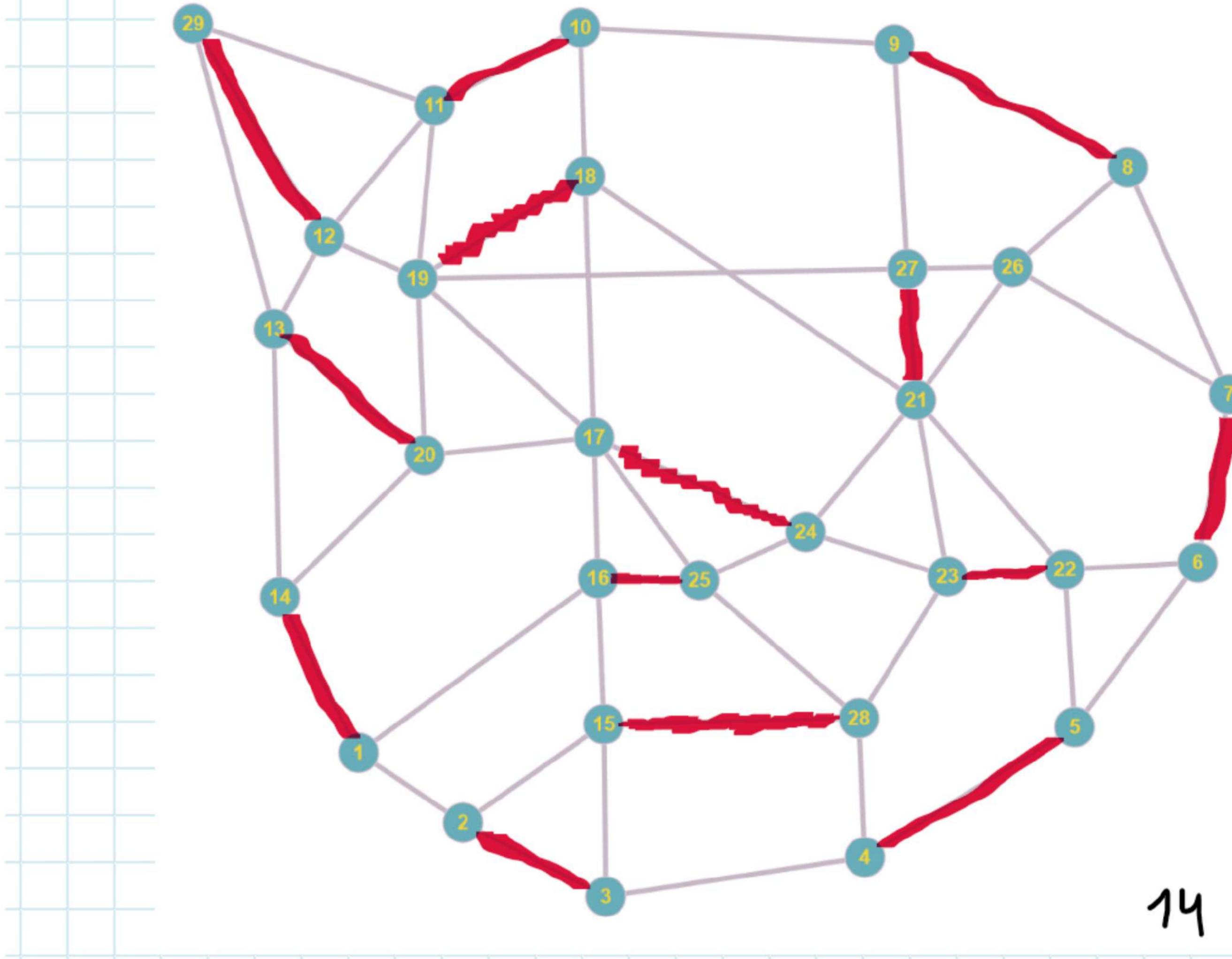
Resulting graph:



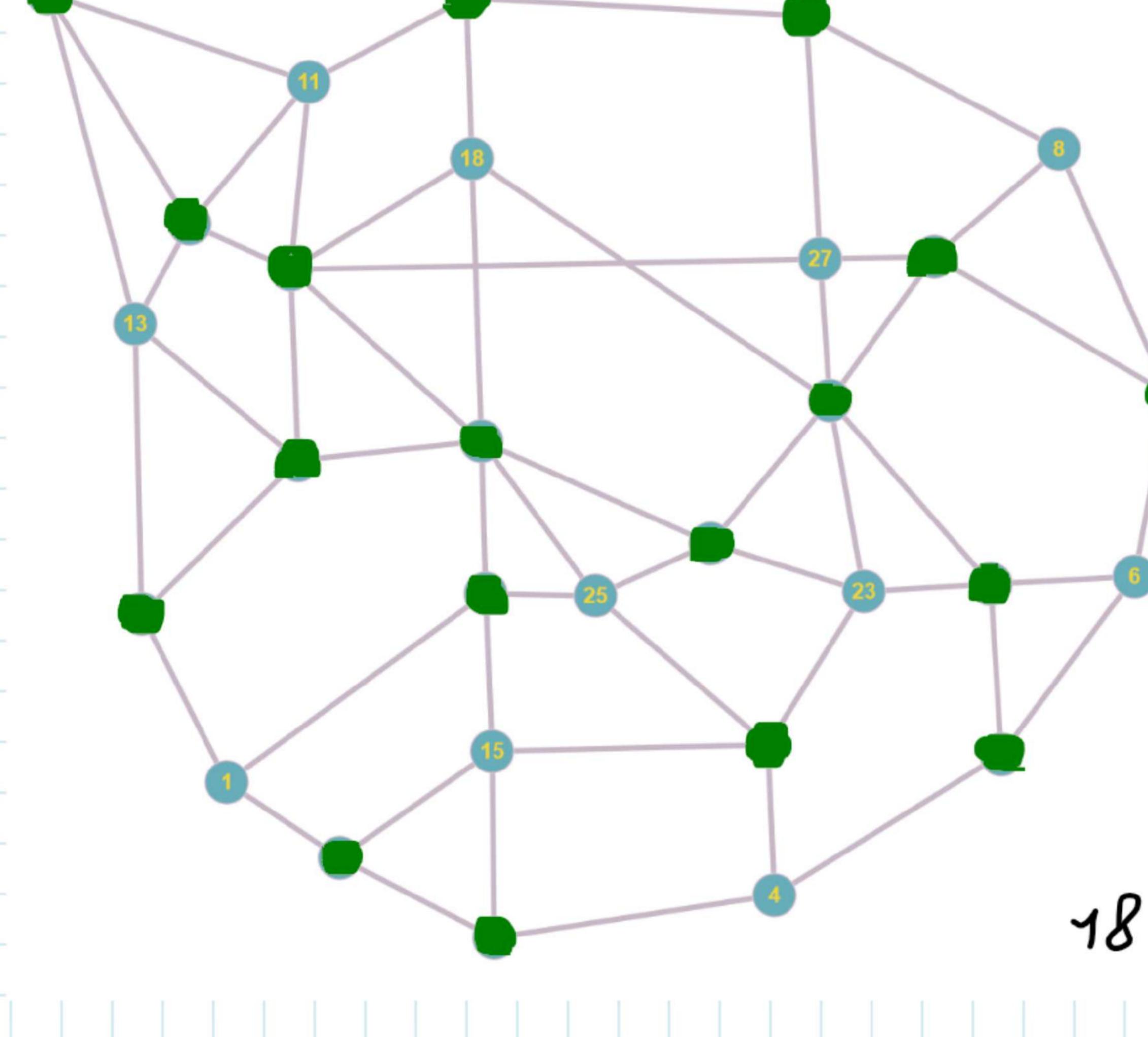
Maximum stable set:



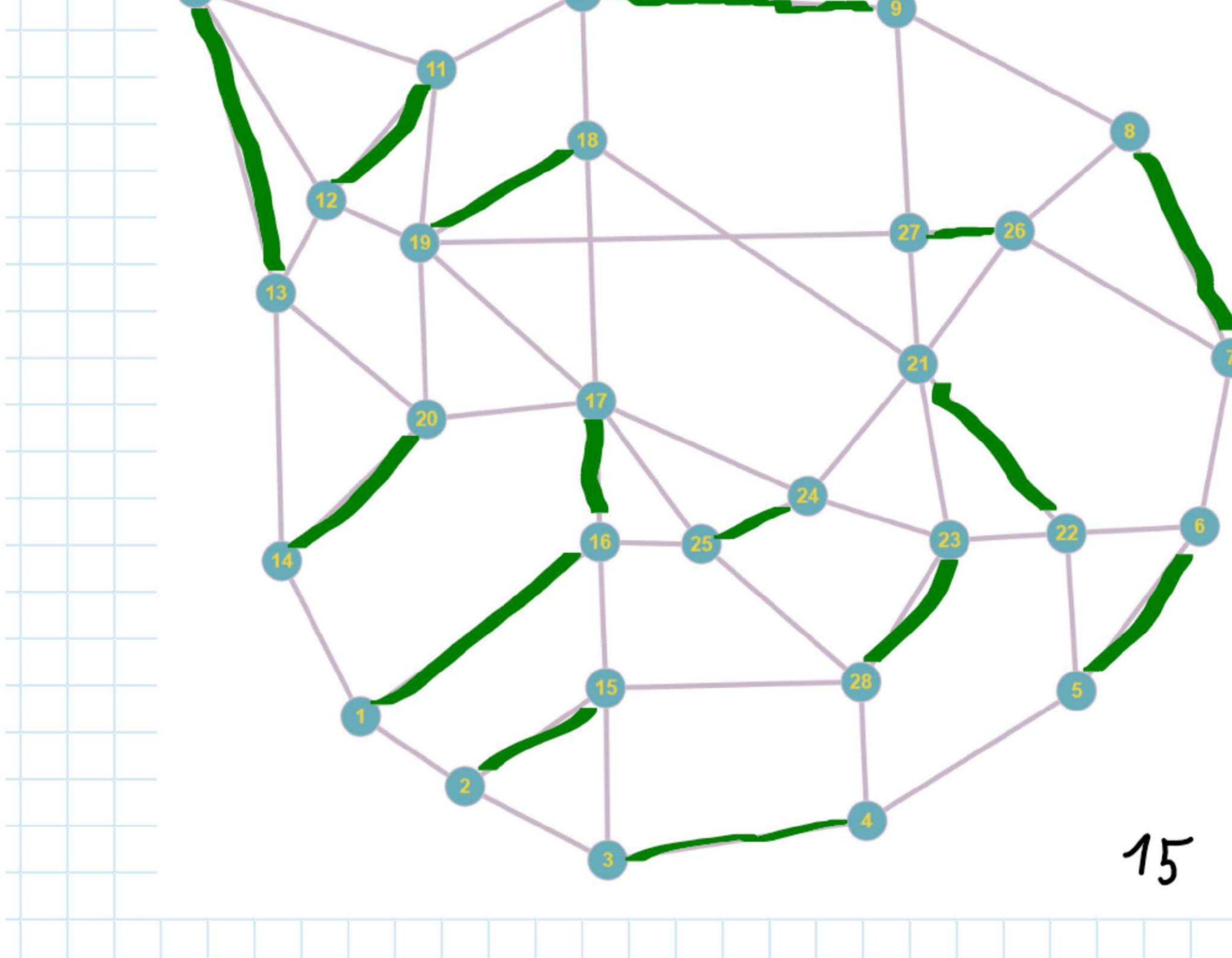
Maximum matching:



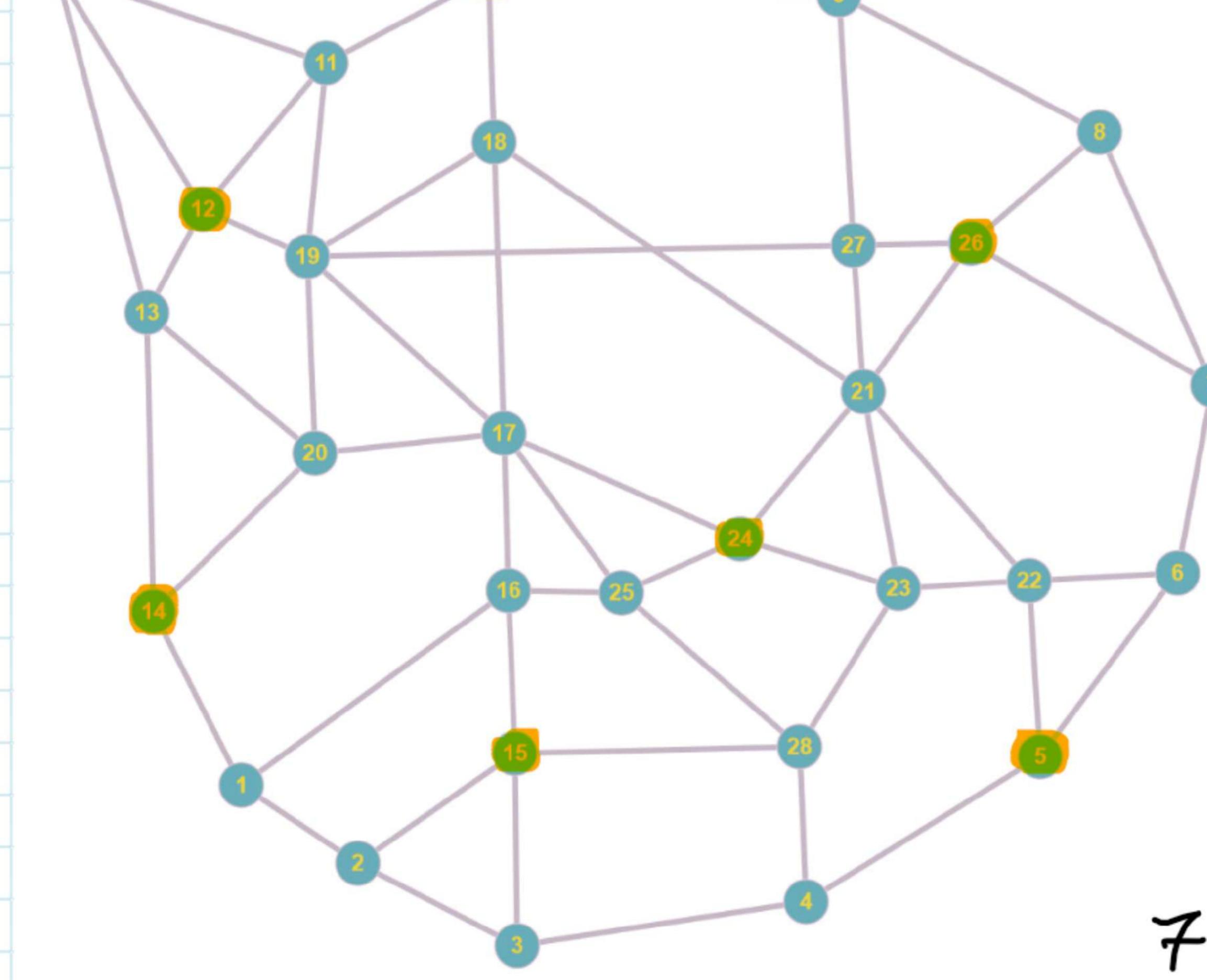
Minimum vertex cover:



Minimum edge cover:



Minimum dominating set:



vertex connectivity - 3

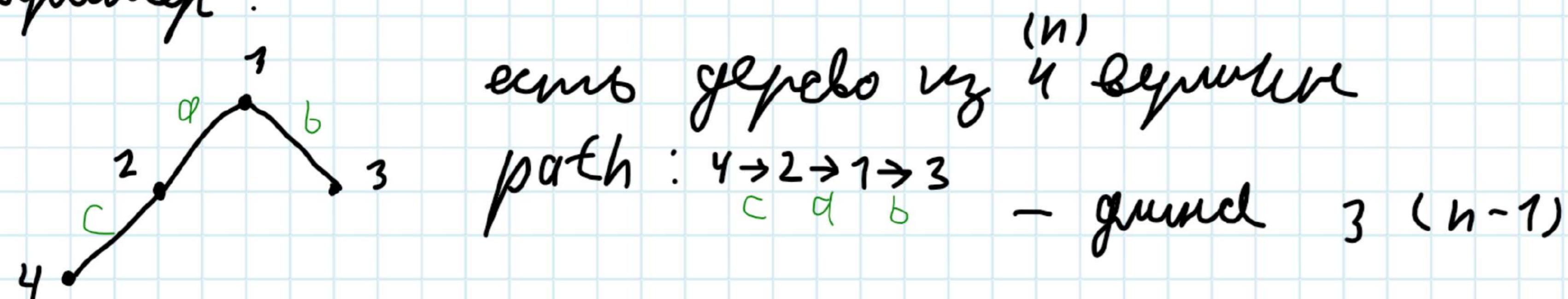
edge connectivity - 3

13. Find an error in the following inductive “proof” of the statement that every tree with n vertices has a path of length $n - 1$.

- Base: A tree with one vertex clearly has a path of length 0. Inductive step: Assume that a tree with n vertices has a path of length $n - 1$, which has u as its terminal vertex. Add a vertex v and the edge from u to v . The resulting tree has $n + 1$ vertices and has a path of length n . □

Данный姑娘инство предполагает, что новую версию для работы в `path`, отдача это не однозначно, поэтому fork-то не нужно.

Гуманітари



То скончано, мы можем доставить новую вершину к 4 шагу 3, в начале пути у нас останется path g_{n-1} , однако мы можем поменять новую вершину к 2, тогда path останется g_{n-2} .

10. Consider the following algorithm (let's call it "Algorithm S") for finding a minimum spanning tree from a connected weighted simple graph $G = \langle V, E \rangle$ by successively adding groups of edges. Suppose that the vertices in V are ordered. Consider the lexicographic order on edges $\langle u, v \rangle \in E$ with $u < v$. An edge $\langle u_1, v_1 \rangle$ precedes $\langle u_2, v_2 \rangle$ if u_1 precedes u_2 or if $u_1 = u_2$ and v_1 precedes v_2 . The algorithm S begins by simultaneously choosing the edge of least weight incident to each vertex. The first edge in the ordering is taken in the case of ties. This produces (you are going to prove it) a graph with no simple circuits, that is, a forest of trees. Next, simultaneously choose for each tree in the forest the shortest edge between a vertex in this tree and a vertex in a different tree. Again, the first edge in the ordering is chosen in the case of ties. This produces an acyclic graph containing fewer trees than before this step. Continue the process of simultaneously adding edges connecting trees until $n - 1$ edges have been chosen. At this stage a minimum spanning tree has been constructed.

 - (a) Show that the addition of edge at each stage of algorithm S produces a forest.
 - (b) Express algorithm S in pseudocode.
 - (c) Use algorithm S to produce a minimum spanning tree for the weighted graph given in task 3.

q) Алгоритм начинается с выбора одновременно ребра с минимальным весом, инцидентного каждой вершине

Поскольку граф является связным взвешенным простым графом по крайней мере, одно ребро инцидентно каждой вершине.

Выбирая ребро с минимальным весом, инцидентное каждой вершине, мы гарантируем, что никакой цикл не будет образован среди этих ребер, поскольку в противном случае существовал бы более короткий путь, нарушающий оптимальность выбранных ребер.

Затем, при выборе ребер между деревьями, алгоритм S выбирает самое короткое ребро между вершиной в одном дереве и вершиной в другом дереве. Снова, выбирая самое короткое ребро, мы гарантируем, что не вводятся циклы.

На каждом этапе алгоритма выбранные ребра добавляются в граф, и циклы не вводятся, потому что ребра выбираются для минимизации веса и избегания образования циклов. Таким образом, граф остается лесом на протяжении выполнения алгоритма.

b) Algorithm S(Graph G):

```
Sort(G.edges, lexicographical)  
forest [];  
mst [];
```

For each vertex v in $G.\text{vertices}$:

```
minEdge = G.edges.findMinIncident(v)
forest.append(minEdge)
mst.append(minEdge)
```

While `forest.size > 1`:

For each tree t in forest:
 $\text{minEdge} = \text{findMinEdge}(t, t.\text{vertices})$
 $t.\text{merge}(\text{minEdge.to})$
 $\text{mst.append}(\text{minEdge})$

Return mst

14. Prove rigorously the following theorems:

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

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

Предположим, что это неверно.

Тогда у нас есть путь из $x \rightarrow z$ через y меньше $\text{dist}(x, z)$, т.к. $\text{dist}(x, z)$ - это определение *крайнейший* путь \Rightarrow противоречие.
Ч.м.д.

Theorem 2. Any connected graph G has $\text{rad}(G) \leq \text{diam}(G) \leq 2 \text{rad}(G)$.

rad - \min радиуса G (но определено)

diam - \max радиуса G (но определено)

$\min \leq \max$, значит $\text{rad}(G) \leq \text{diam}(G)$ верно

Пусть V - vertex с \max радиусом, т.к. G - connected есть vertex u такой, что $\text{dist}(v, u) = \text{diam}(G)$, а $\text{dist}(u, \text{any vertex}) \leq \text{rad}(G)$

Покажем $\text{dist}(v, u) = \text{diam}(G)$, а $\text{dist}(u, \text{any vertex}) \leq \text{rad}(G)$

Предположим $\text{diam}(G) \leq 2 \text{rad}(G)$. Докажем

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

Part 1: If a connected graph $G = \langle V, E \rangle$ is a tree, then $|E| = |V| - 1$.

Доказательство по индукции по количеству вершин, $|V|$:

База: Для $|V|=1$, $|E|=0$, граф имеет только одну вершину, и тогда очевидно, что $|E|=|V|-1=1-1=0$.

Переход: Предположим, что утверждение верно для всех деревьев с n вершинами, где $n > 1$. Теперь рассмотрим дерево T с $n+1$ вершинами.

Поскольку T является деревом, оно связно и ациклическо. Удалим любую листовую вершину (вершину степени 1) из T . Это удаление не отсекает граф, поскольку это лист, и оставшийся граф все еще будет связным. Пусть T' будет полученным деревом.

Теперь у T' есть n вершин и $|E'| = |V'| - 1$ по предположению индукции.

Поскольку мы удалили только одну вершину и ее инцидентное ребро, $|E| = |E'| + 1$, а $|V| = |V'| + 1$. Следовательно,

$$|E| = |E'| + 1 = |V'| - 1 + 1 = |V| - 1$$

Таким образом, теорема верна для случая $n+1$.

Part 2: If $|E| = |V| - 1$, then $G = \langle V, E \rangle$ is a tree.

Доказательство от противного:

Предположим, что G не является деревом, что означает, что в нем есть цикл C . Тогда, для любого цикла в G , удаление любого ребра в цикле не разъединит граф, так как имеются альтернативные пути.

Пусть ее будет любым ребром в цикле C . Удаление ее из G не разъединит G , поэтому G остается связным.

Поскольку G остается связным после удаления ее, $|E|$ остается таким же, но $|V|$ уменьшается на 1.

$$|E| = |V| - 1$$

$$|E| = (|V| - 1) - 1$$

$$|E| = |V| - 2$$

Это противоречит предположению о том, что $|E| = |V| - 1$. Следовательно, наше предположение о том, что G содержит цикл, должно быть ложным.

Таким образом, если $|E| = |V| - 1$, то G ациклическо, то есть является деревом.

Обе части теоремы доказаны, устанавливая эквивалентность между связным графом, являющимся деревом, и $|E| = |V| - 1$.

Theorem 4 (WHITNEY). For any graph G : $\kappa(G) \leq \lambda(G) \leq \delta(G)$.

Если в graph есть bridge, то из него достаточно удалить vertex смежную с мостом, чтобы graph перестал быть connected. Если удалить из graph $\lambda(G)-1$ ребро, то в нем останется bridge. Значит $\kappa(G) \leq \lambda(G)$ верно

Если из graph удалить edges смежные с вершиной со степенью $\delta(G)$ то graph перестает быть connected, значит $\lambda(G) \leq \delta(G)$ верно
и.m.g.

Theorem 5 (CHARTRAND). For a connected graph $G = \langle V, E \rangle$: if $\delta(G) \geq \lfloor |V|/2 \rfloor$, then $\lambda(G) = \delta(G)$.

$\lambda(G)$ разбиваем graph на connectivity components, пусть меньшая из них M , тогда $|M| \leq \frac{|V|}{2}$. Every vertex in M соединена как минимум с $\delta(G)$ vertices, значит у каждой vertex из M хотя бы $\delta(G)-|M|+1$ соседей из $V \setminus M$, значит

$$\lambda(G) \geq |M| * (\delta(G) - |M| + 1), \quad 1 \leq |M| \leq \frac{|V|}{2}$$

by Whitney theorem мы знаем, что $\lambda(G) \leq \delta(G)$ $\Rightarrow \lambda(G) = \delta(G)$

Theorem 6 (HARARY). Every block of a block graph² is a clique.

Пусть G - блок-граф, а B - блок в G .

Чтобы показать, что B является clique, нам нужно доказать, что каждая пара vertices в B смежна.

Предположим, для противоречия, что существуют две vertices u и v в B , которые не смежны.

Поскольку B - блок, существует путь между u и v в B , иначе добавление edge uv создало бы более крупный connected подграф без разрезных vertices, что противоречит максимальности B .

Теперь рассмотрим vertices на этом пути между u и v . Поскольку u и v не смежны, должна быть хотя бы одна vertex, скажем, w , на этом пути, которая не смежна ни с u , ни с v .

Но это означает, что w является разрезной vertex в блоке B , что противоречит определению блока как максимального connected подграфа без разрезных вершин.

Следовательно, наше предположение о том, что существуют две несмежные vertices в B , неверно. Таким образом, каждая пара vertices в B должна быть смежна.

Следовательно, B является clique.

Поскольку B был произвольным блоком в блок-графе G , мы показали, что каждый блок G является clique.