DM1 GAVRILOV MISHA

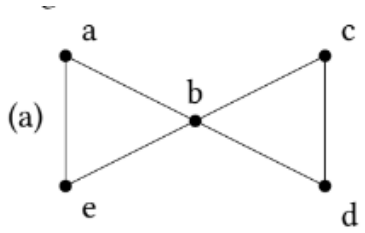
PS. Я не знаю почему разметка страниц такая странная при конвертации файла notion в pdf. Поэтому советую смотреть решения тутъ(там все достаточно красивенько и нет супер-длинных отступов 😟)

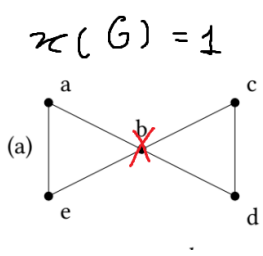
Task 1

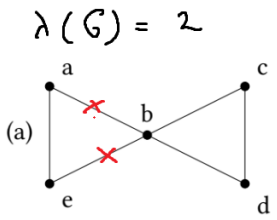
---

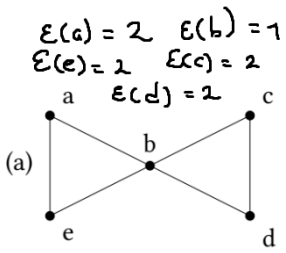
1. For each of the following graphs, find $\varkappa(G)$, $\lambda(G)$, $\delta(G)$, $\varepsilon(v)$ of each vertex $v \in V(G)$, rad(G), diam(G), center(G). Find Euler path, Euler circuit, and Hamiltonian cycle, if they exist. In addition, find maximum clique $Q \subseteq V$, maximum stable set $S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

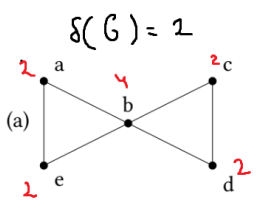
a)

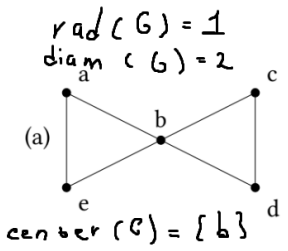












Euler path - a→b→c→d→b→e→a

Euler circuit - a→b→c→d→b→e→a

Hamiltonian cycle - **does not exist**

Maximum clique - {a, b, e}

Maximum stable set S ⊆ V - {a, c}

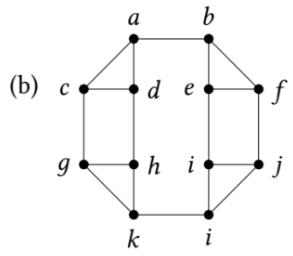
Maximum matching M ⊆ E - {a↔e, c↔d}

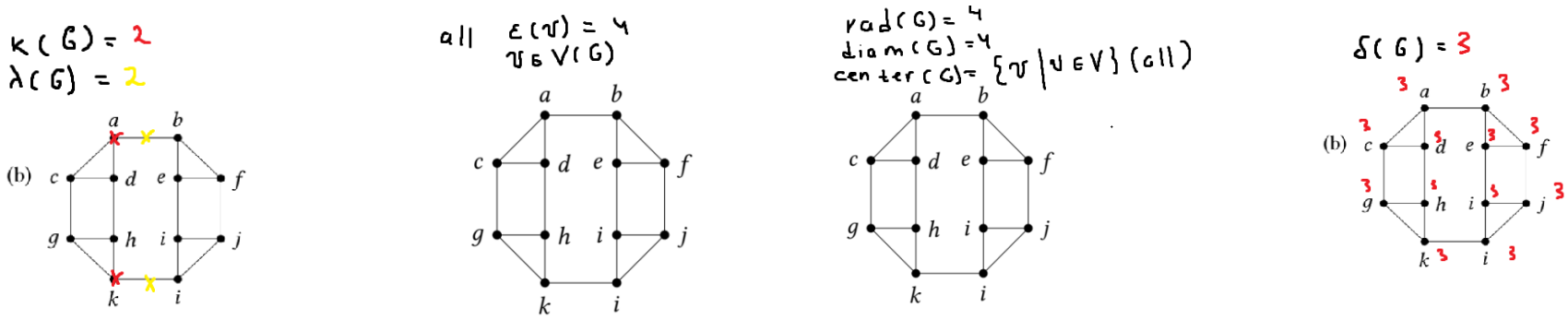
Minimum dominating set D ⊆ V - {b}

Minimum vertex cover R ⊆ V - {a, b, c}

Minimum edge cover F ⊆ E of G - {a↔b, e↔b, c↔d}

b)





Euler path - **does not exist**

Euler circuit - **does not exist**

Hamiltonian cycle - a → c → d → h → g → k → l → i → j → f → e → b → a

Maximum clique - {a, c, d}

Maximum stable set S ⊆ V - {b, c, h, j}

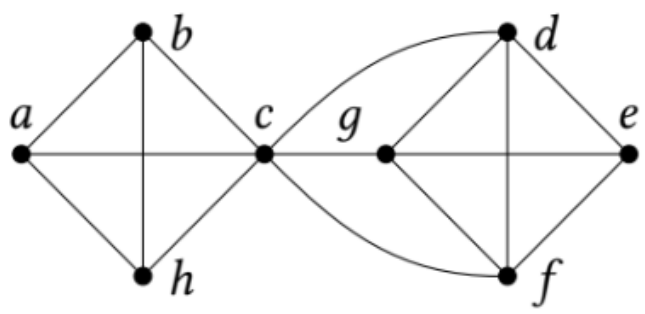
Maximum matching M ⊆ E - {a↔c, d↔h, g↔k, l↔j, i↔e, b↔f}

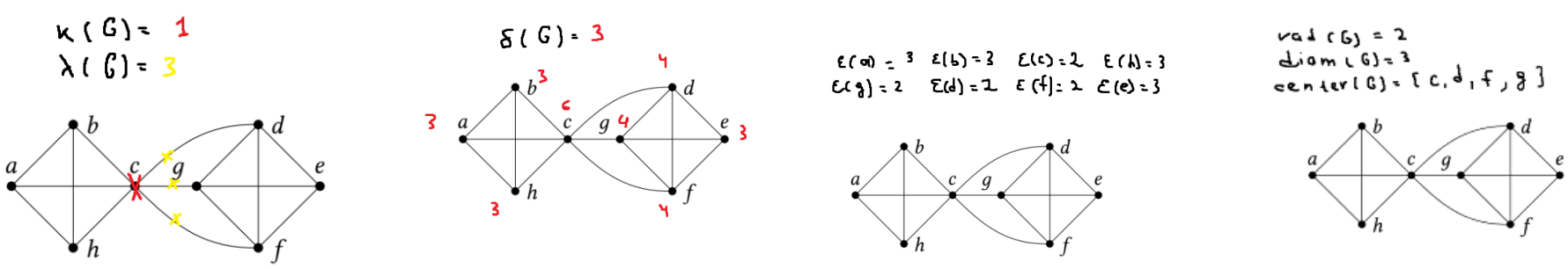
Minimum dominating set D ⊆ V - {a, b, h, i}

Minimum vertex cover R ⊆ V - {b, c, d, f, g, i, j, k}

Minimum edge cover F ⊆ E of G - {a↔c, d↔h, g↔k, l↔j, i↔e, b↔f}

c)





Euler path - **does not exist**

Euler circuit - **does not exist**

Hamiltonian cycle - **does not exist**

Maximum clique - {a, b, c, h}

Maximum stable set S ⊆ V - {a, e}

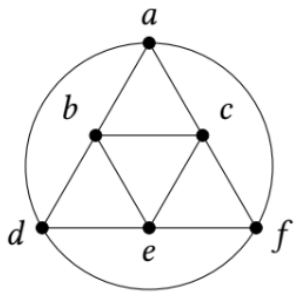
Maximum matching M ⊆ E - {a↔b, c↔h, g↔d, e↔f}

Minimum dominating set D ⊆ V - {c, g}

Minimum vertex cover R ⊆ V - {a, c, h, d, e, f}

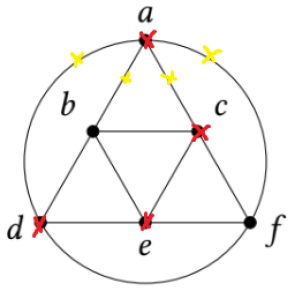
Minimum edge cover F ⊆ E of G - {a↔b, c↔h, g↔d, e↔f}

d)

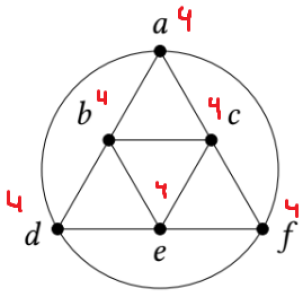


$\kappa(G) = 4$

$\lambda(G) = 4$

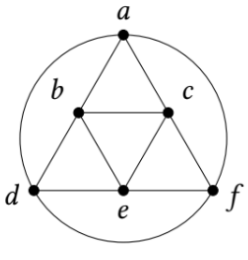


$\delta(G) = 4$



$\varepsilon(a) = 2 \quad \varepsilon(b) = 2 \quad \varepsilon(c) = 2$

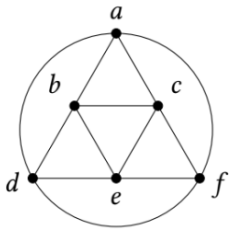
$\varepsilon(d) = 2 \quad \varepsilon(e) = 2 \quad \varepsilon(f) = 2$



$rad(G) = 2$

$diam(G) = 2$

$center = \{v \mid v \in V(G)\}$ (all)



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

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

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

Maximum clique - {b, c, e}

Maximum stable set S ⊆ V - {d, c}

Maximum matching M ⊆ E - {a↔b, c↔e, d↔f}

Minimum dominating set D ⊆ V - {a, e}

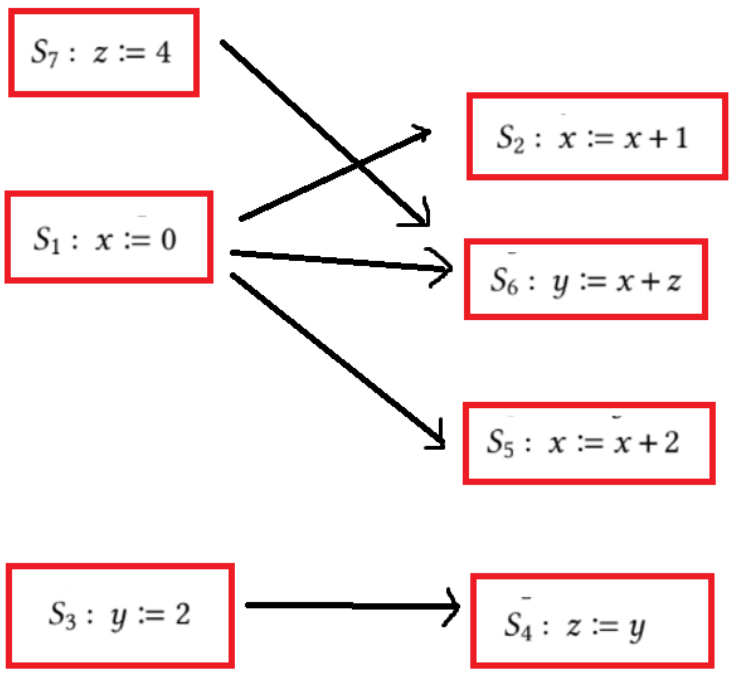
Minimum vertex cover R ⊆ V - {a, c, d, e}

Minimum edge cover F ⊆ E of G - {a↔d, b↔e, c↔f}

Task 2

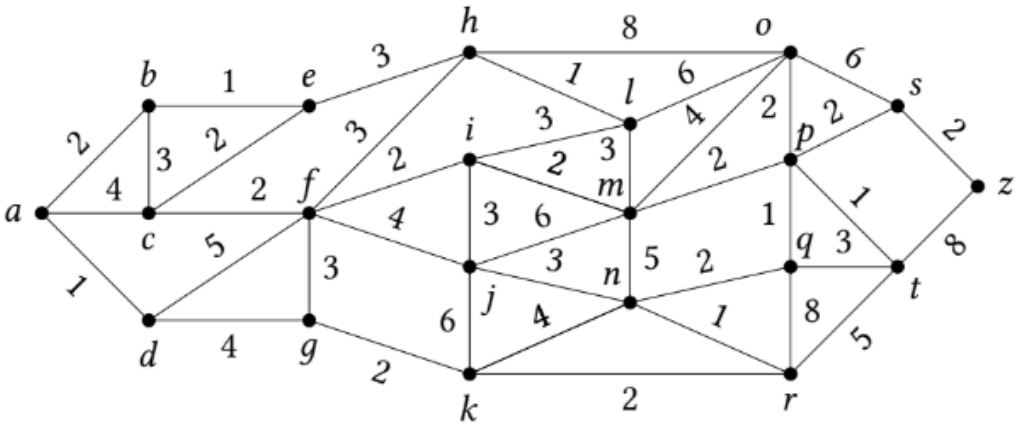
A **precedence graph** is a directed graph where the vertices represent the program instructions and the edges represent the dependencies between instructions: there is an edge from one statement to a second statement if the second statement cannot be executed before the first statement.
For example, 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:

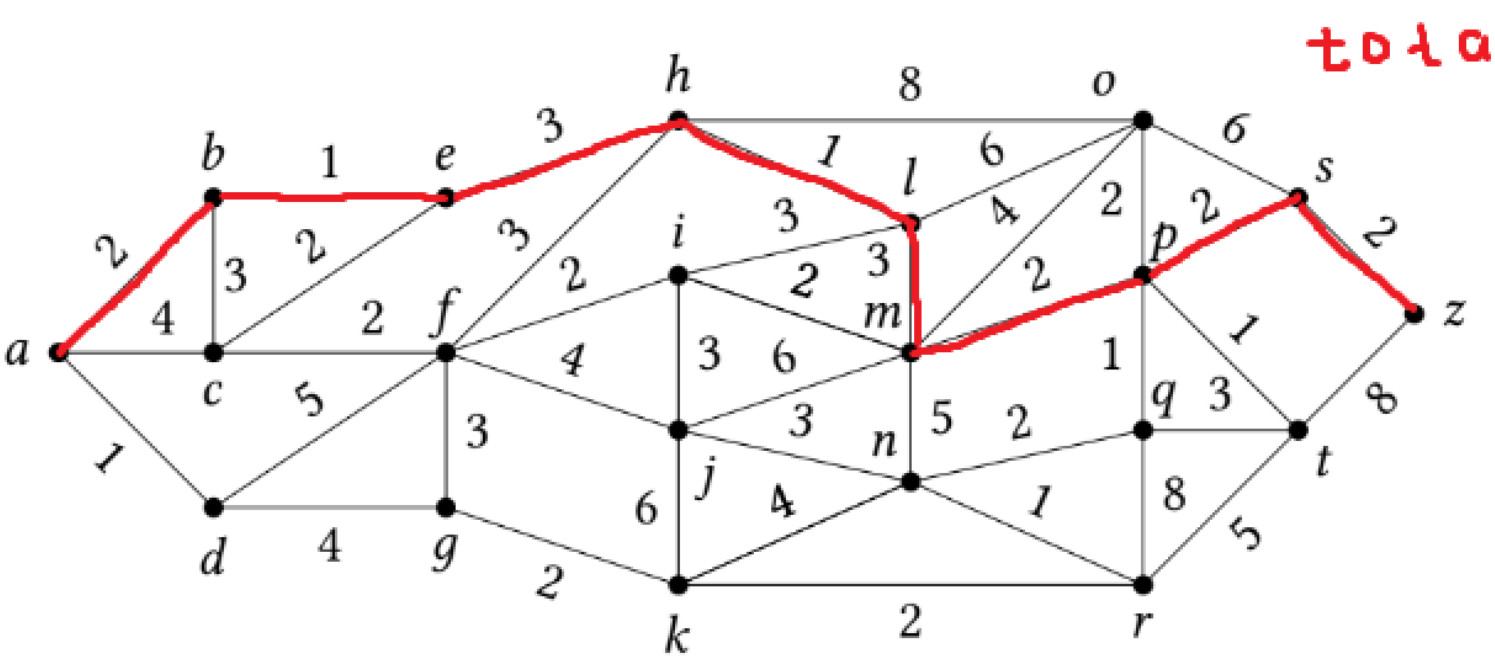
$S_1 : x := 0$
$S_2 : x := x + 1$
$S_3 : y := 2$
$S_4 : z := y$
$S_5 : x := x + 2$
$S_6 : y := x + z$
$S_7 : z := 4$



Task 3

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





Code that proves this

```
#include <iostream>
#include <vector>
#include <queue>


const int kInf = 1e9;
std::vector<std::vector<std::pair<int,int>>> v(26);


std::vector<int> dijkstra(std::vector<std::vector<std::pair<int,int>>>& graph, int start) {
    std::vector<int> dist(graph.size(), kInf);
```

```
dist[start] = 0;
std::priority_queue<std::pair<int,int>, std::vector<std::pair<int,int>>, std::greater<>> q;
q.emplace(dist[start], start);

while (!q.empty()) {
    auto [nearest, near] = q.top();
    q.pop();

    if (nearest != dist[near])
        continue;

    for (auto& [to, weight] : graph[near]) {
        if (dist[to] > dist[near] + weight) {
            dist[to] = dist[near] + weight;
            q.emplace(dist[to], to);
        }
    }
}

return dist;
}

void AddEdge(char a, char b, int t) {

    int st = a - 97;
    int fin = b - 97;

    v[st].emplace_back(fin, t);
    v[fin].emplace_back(st, t);

}

int main() {

    AddEdge('a', 'b', 2);
    AddEdge('a', 'c', 4);
    AddEdge('a', 'd', 1);
    AddEdge('b', 'e', 1);
    AddEdge('b', 'c', 3);
    AddEdge('c', 'e', 2);
    AddEdge('c', 'f', 2);
    AddEdge('d', 'f', 5);
    AddEdge('d', 'g', 4);
    AddEdge('f', 'g', 3);
    AddEdge('e', 'h', 3);
    AddEdge('f', 'h', 3);
    AddEdge('g', 'k', 2);
    AddEdge('f', 'j', 4);
    AddEdge('f', 'i', 2);
    AddEdge('i', 'j', 3);
    AddEdge('j', 'k', 6);
    AddEdge('h', 'o', 8);
    AddEdge('h', 'l', 1);
    AddEdge('i', 'l', 3);
    AddEdge('i', 'm', 2);
    AddEdge('j', 'm', 6);
    AddEdge('j', 'n', 3);
    AddEdge('k', 'n', 4);
    AddEdge('k', 'r', 2);
    AddEdge('l', 'm', 3);
    AddEdge('m', 'n', 5);
    AddEdge('l', 'o', 6);
    AddEdge('m', 'o', 4);
    AddEdge('m', 'p', 2);
    AddEdge('n', 'q', 2);
    AddEdge('n', 'r', 1);
    AddEdge('o', 'p', 2);
```

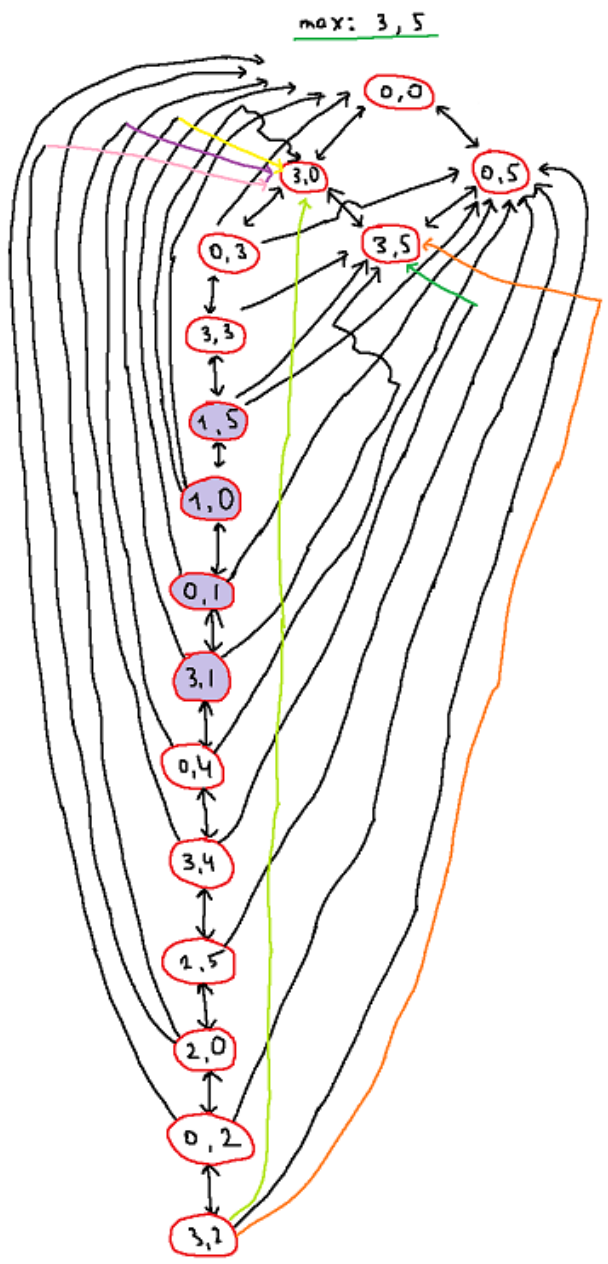
```
    AddEdge('p', 'q', 1);
    AddEdge('q', 'r', 8);
    AddEdge('o', 's', 6);
    AddEdge('p', 's', 2);
    AddEdge('p', 't', 1);
    AddEdge('q', 't', 3);
    AddEdge('r', 't', 5);
    AddEdge('s', 'z', 2);
    AddEdge('t', 'z', 8);

    std::vector<int> res = dijkstra(v, 0);

    std::cout << (res[res.size() - 1] != kInf ? res[res.size() - 1] : -1) << '\n';
    return 0;
}
```

Task 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.

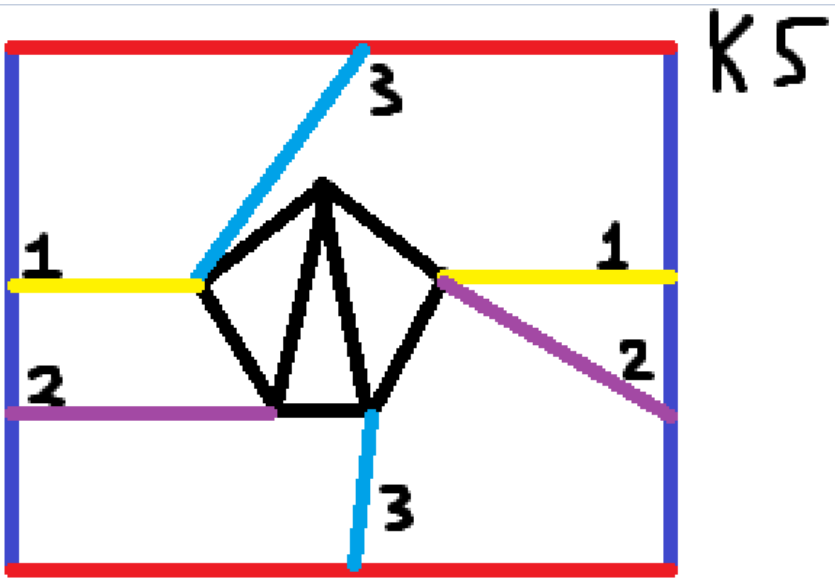


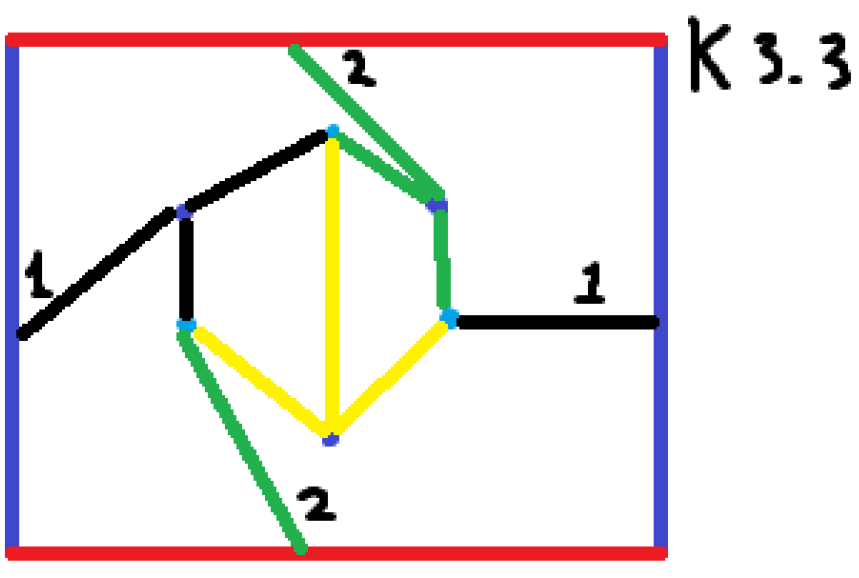
Пусть в вершинах графа будут пары (a, b), где a - литров в 3-ой банке, а b - литров в 5 - ой банке, направленное ребро - какая-либо операция с банками(опустошить 3-ую банку, опустошить 5-ую банку, перелить из 3-ой в 5-ую, перелить из 5-ой в 3-ую).

P.S некоторые ребра покрасил в какой-то цвет, чтоб картинка слишком не сливалась.

Task 5

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





Task 6

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.

My C++ code

```
#include <iostream>
#include <vector>
#include <algorithm>


const int kInf = 1e9;

std::vector<std::vector<int>> dist(26, std::vector<int>(26, kInf));

std::vector<std::vector<int>> from(26, std::vector<int>(26, -1));


void AddEdge(char a, char b, int t) {
    int st = a - 97;
    int fin = b - 97;

    if (dist[st][fin] > t) {
        dist[st][fin] = t;
        from[st][fin] = st;
    }

    if (dist[fin][st] > t) {
        dist[fin][st] = t;
        from[fin][st] = fin;
    }

}

void InitGraph() {
    AddEdge('a', 'b', 2);
    AddEdge('a', 'c', 4);
    AddEdge('a', 'd', 1);
    AddEdge('b', 'e', 1);
    AddEdge('b', 'c', 3);
```

```
    AddEdge('c', 'e', 2);
    AddEdge('c', 'f', 2);
    AddEdge('d', 'f', 5);
    AddEdge('d', 'g', 4);
    AddEdge('f', 'g', 3);
    AddEdge('e', 'h', 3);
    AddEdge('f', 'h', 3);
    AddEdge('g', 'k', 2);
    AddEdge('f', 'j', 4);
    AddEdge('f', 'i', 2);
    AddEdge('i', 'j', 3);
    AddEdge('j', 'k', 6);
    AddEdge('h', 'o', 8);
    AddEdge('h', 'l', 1);
    AddEdge('i', 'l', 3);
    AddEdge('i', 'm', 2);
    AddEdge('j', 'm', 6);
    AddEdge('j', 'n', 3);
    AddEdge('k', 'n', 4);
    AddEdge('k', 'r', 2);
    AddEdge('l', 'm', 3);
    AddEdge('m', 'n', 5);
    AddEdge('l', 'o', 6);
    AddEdge('m', 'o', 4);
    AddEdge('m', 'p', 2);
    AddEdge('n', 'q', 2);
    AddEdge('n', 'r', 1);
    AddEdge('o', 'p', 2);
    AddEdge('p', 'q', 1);
    AddEdge('q', 'r', 8);
    AddEdge('o', 's', 6);
    AddEdge('p', 's', 2);
    AddEdge('p', 't', 1);
    AddEdge('q', 't', 3);
    AddEdge('r', 't', 5);
    AddEdge('s', 'z', 2);
    AddEdge('t', 'z', 8);
}

std::vector<char> GetPath(char st_char, char fin_char) {

    int st = st_char - 97;
    int fin = fin_char - 97;

    std::vector<char> path;

    for (int i = fin; i != -1; i = from[st][i]) {
        path.push_back(static_cast<char>(i + 97));
    }

    std::reverse(path.begin(), path.end());

    return path;
}

int main() {

    for (size_t i = 0; i < 26; ++i) {
        dist[i][i] = 0;
    }

    InitGraph();

    for (size_t v = 0; v < 26; ++v) {
        for (size_t a = 0; a < 26; ++a) {
            for (size_t b = 0; b < 26; ++b) {
                if (dist[a][v] != kInf && dist[v][b] != kInf &&
                    dist[a][b] > dist[a][v] + dist[v][b]) {
```

```
                dist[a][b] = dist[a][v] + dist[v][b];
                from[a][b] = from[v][b];

            }

        }
    }
}

for (char i = 'a'; i <= 'z'; ++i) {

    for (char j = 'a'; j <= 'z'; ++j) {
        std::cout << "Start vert - " << i << " End vert - " << j << " Current path : ";
        for (auto v : GetPath(i, j)) {
            std::cout << v << " ";
        }

        std::cout << " Distance = " << dist[i - 97][j - 97];
        std::cout << '\n';
    }

    return 0;
}
```

```
Start vert - a End vert - z Current path : a c f i m p s z  Distance = 16
```

Пример вывода программы.

a)

(a) 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.

The code is at the top

b)

(b) Prove that Floyd's algorithm determines the shortest distance between all pairs of vertices in a weighted simple graph.

Обозначим за $dist[k, i, j]$ - длина минимального пути из i в j с вершинами из множества $[0, k)$. Рассмотрим динамику :

База : k равно 0

Если i и j - смежные, то $dist[0, i, j] = l_{i,j}$

Если i и j - не смежные, то $dist[0, i, j]) = +\infty$

Переход :

Будем пытаться релаксировать $i \rightsquigarrow j$, через вершину k.

$dist[k + 1, i, j] = \min(dist[k, i, j], dist[k, i, k] + dist[k, k, j])\ \forall i, j$

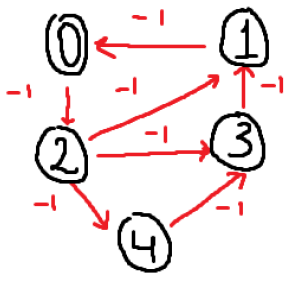
Результат :

Ответ хранится в ячейке $dist[n - 1, i, j]$, так как мы рассматриваем пути с вершинами из множества $[0, n)$.

Данная динамика верная, а алгоритм Флойда следует из этой динамики, так как нам достаточно двумерного массива без измерения k.

c)

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).



	0	1	2	3	4
0	0	∞	-1	∞	∞
1	-1	0	∞	∞	∞
2	∞	-1	0	-1	-1
3	∞	-1	∞	0	∞
4	∞	∞	∞	-1	0

Пусть у нас был такой граф и матрица $dist$ после инициализацации.

После работы алгоритма Флойда мы получаем результирующую матрицу $dist$.

	0	1	2	3	4
0	-62	-61	-63	-70	-113
1	-63	-62	-64	-71	-113
2	-64	-63	-65	-72	-120
3	-73	-72	-74	-81	-129
4	-106	-105	-107	-114	-162

Мы получаем такой результат, потому что постоянно можем релаксировать путь через отрицательный цикл и уменьшать его. При чем мы можем получать неверный ответ не только для вершин на отрицательном цикле, но и для вершин, достижимых из них.

d)

(d) 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.

Algorithm 1: Floyd's algorithm

Data: weighted simple graph $G = \langle V, E, w \rangle$ with vertices $V = \{v_1, \ldots, v_n\}$ and weights $w(v_i, v_j)$, where $w(v_i, v_j) = \infty$ if $\langle v_i, v_j \rangle \notin E$.
Result: $d(v_i, v_j)$ is the length of a shortest path between v_i and v_j.

```
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)
4 for i := 1 to n do
5     for j := 1 to n do
6         for k := 1 to n do
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)
```

Total operations : $n^2 + n^3$

Result : $O(n^3)$

e)

(e) Modify the algorithm to output the actual shortest path between any two given vertices, not just the distance between them.

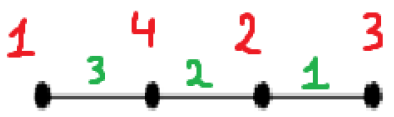
The code is at the top

Task 7

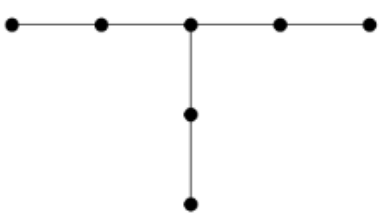
A tree with n vertices is called **graceful** if its vertices can be labeled with the integers $1, 2, \ldots, n$ in such a way that the absolute values of the difference of the labels of adjacent vertices are all different. Show that the following graphs are graceful.

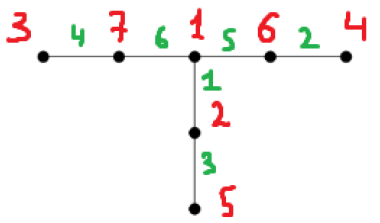
a)

b)

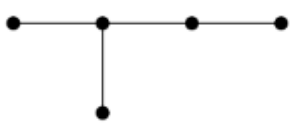


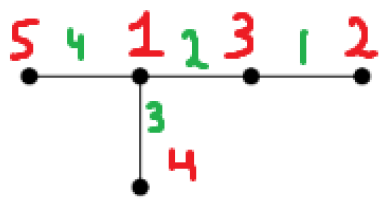
b)



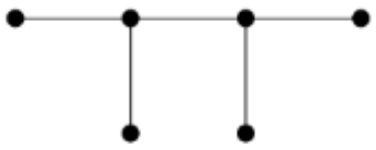


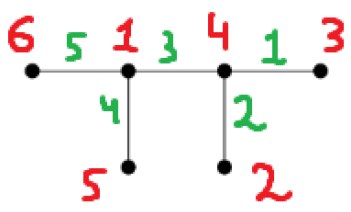
c)





d)





Task 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?

Caterpillars : a, c, d (по определению)

b)

How many non-isomorphic caterpillars are there with six vertices?

Рассмотрим каждую максимальную длину пути содержащегося в $caterpillar$.

Будем вести $counter$ возможных вариантов.

5(6 vertex) : Этот путь уже является $caterpillar \rightarrow counter++$

4(5 vertex) : У нас остается 1 лишняя вершина, которую мы можем подвесить за каждую, кроме начальной и конечной(иначе выродится в путь длины 5). Но если мы подвесим за 2 или 4 вершину, то получим изоморфные графы. Значит всего 2 способа. $counter+=2$

3(4 vertex) : У нас остается 2 лишние вершины, которые мы можем подвесить к 2 или 3 вершине(иначе выродится в путь длины 4). Тут есть 2 неизоморфные гусеницы(подвешено к 2 и 3, подвешено только к 2 или 3 вершинам). $counter=+=2$.

2 (3 vertex) : У нас остается 3 лишние вершины, которые можем подвесить только к 2 вершине. То есть существует только 1 неизоморфная гусеница. $counter+=1$

1 (2 vertex) : У нас нет вершин между начальной и конечной, поэтому для данного случая $counter+=0$

Общее количество : $counter = 6$

c)

Prove or disprove that all caterpillars are graceful.

Пусть $caterpillar = C$ и имеет N вершин, а также путь максимальной длины равен K. Докажем что $C - graceful$.

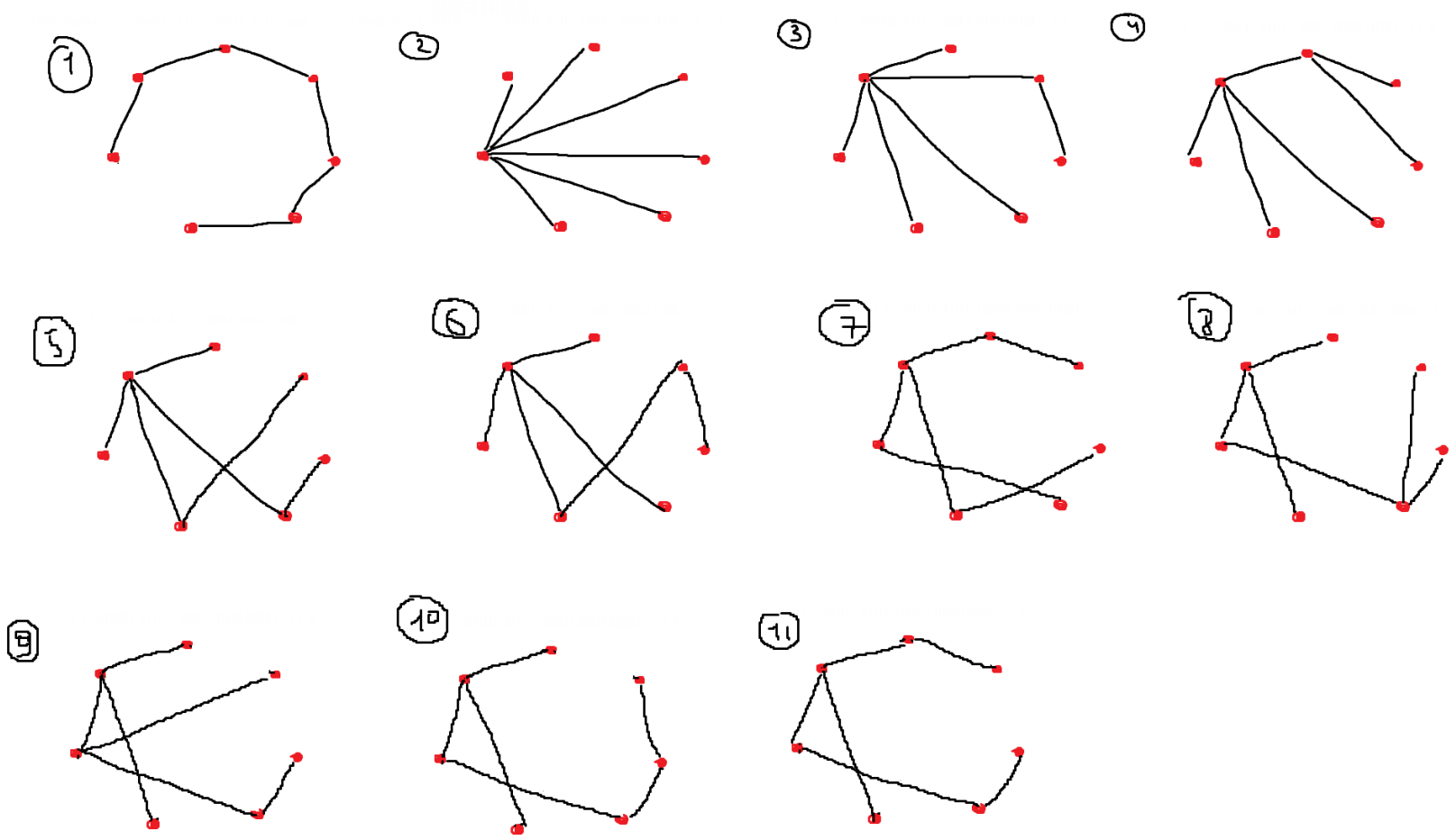
Отметим 1 вершину пути - 1, а смежную с ней на пути - n, всех ей детей будем помечать последовательными числами от $2 \dots k$. И смежную вершину с вершиной n в пути, отметим как $k+1$, а всех ее детей будем отмечать от $n-1, n-2, \dots, m$. Смежную вершину с $k+1$ в пути отметим, как $m-1$.

И так сделаем по всем остальным вершинам пути.

Тогда разница между соседними будет равна $n-1 \dots 2, 1$ и все соседние вершины различны.

Task 9

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



Task 10

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 $u_1 = u_1$ 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)

(a) Show that the addition of edge at each stage of algorithm S produces a forest.

Переформулируем : докажем, что при добавлении ребра не образуется циклов.

Пусть цикл есть. Тогда выберем на нем какие-то две последовательные вершины. Допустим ребро между ними попало в остов, так как было минимальным для первой вершины, тогда другое ребро, инцидентное второй вершине должно быть меньше него, чтобы попасть в остов. Далее сделаем аналогичные рассуждения для всех последовательных вершин во всем цикле. Однако мы придем к противоречию, так как первое ребро должно быть максимальным из них, но последнее получится больше него.

b)

(b) Express algorithm S in pseudocode.

C++ CODE :

```
#include <iostream>
#include <vector>


struct Edge {
    Edge(int s, int d, int w) : src(s), dest(d), weight(w) {}
    Edge() = default;

    int src, dest, weight;
};

std::vector<Edge> edges;

void AddEdge(char a, char b, int t) {

    int st = a - 97;
    int fin = b - 97;

    edges.emplace_back(st, fin, t);
    edges.emplace_back(fin, st, t);
}

void InitGraph() {
    edges.resize(26);

    AddEdge('a', 'b', 2);
    AddEdge('a', 'c', 4);
    AddEdge('a', 'd', 1);
    AddEdge('b', 'e', 1);
    AddEdge('b', 'c', 3);
    AddEdge('c', 'e', 2);
    AddEdge('c', 'f', 2);
    AddEdge('d', 'f', 5);
    AddEdge('d', 'g', 4);
    AddEdge('f', 'g', 3);
    AddEdge('e', 'h', 3);
    AddEdge('f', 'h', 3);
    AddEdge('g', 'k', 2);
    AddEdge('f', 'j', 4);
    AddEdge('f', 'i', 2);
    AddEdge('i', 'j', 3);
    AddEdge('j', 'k', 6);
    AddEdge('h', 'o', 8);
    AddEdge('h', 'l', 1);
    AddEdge('i', 'l', 3);
    AddEdge('i', 'm', 2);
    AddEdge('j', 'm', 6);
    AddEdge('j', 'n', 3);
    AddEdge('k', 'n', 4);
    AddEdge('k', 'r', 2);
    AddEdge('l', 'm', 3);
```

```
    AddEdge('m', 'n', 5);
    AddEdge('l', 'o', 6);
    AddEdge('m', 'o', 4);
    AddEdge('m', 'p', 2);
    AddEdge('n', 'q', 2);
    AddEdge('n', 'r', 1);
    AddEdge('o', 'p', 2);
    AddEdge('p', 'q', 1);
    AddEdge('q', 'r', 8);
    AddEdge('o', 's', 6);
    AddEdge('p', 's', 2);
    AddEdge('p', 't', 1);
    AddEdge('q', 't', 3);
    AddEdge('r', 't', 5);
    AddEdge('s', 'z', 2);
    AddEdge('t', 'z', 8);
}

//Algo on DSU
struct Subset {
    int parent, rank;
};

// with heuristic
int find(std::vector<Subset>& subsets, int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }

    return subsets[i].parent;
}

// with heuristic
void UnionSets(std::vector<Subset>& subsets, int x, int y) {
    int x_root = find(subsets, x);
    int y_root = find(subsets, y);

    if (subsets[x_root].rank > subsets[y_root].rank) {
        std::swap(subsets[x_root], subsets[y_root]);
    }

    if (subsets[x_root].rank == subsets[y_root].rank) {
        subsets[x_root].rank++;
    }

    subsets[x_root].parent = y_root;
}

void OutData(std::vector<Edge>& selected, int cur_weight) {
    std::cout << "MST Weight: " << cur_weight << '\n';
    std::cout << "Selected Edges:" << '\n';
    for (const auto& edge : selected) {
        std::cout << static_cast<char>(edge.src + 97) << " -- " << static_cast<char>(edge.dest + 97) << " Weight:
    }
}

void Algo_S(int V) {
    std::vector<Edge> selected;

    std::vector<Subset> subsets(V);
    std::vector<int> cheapest(V);

    int tree_counter = V;
    int cur_weight = 0;

    //Base
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
```

```
        subsets[v].rank = 0;
        cheapest[v] = -1;
    }

    // Combining components until all components are in one tree without : x, y, u, v, w = 5 + 1 = 6
    while (tree_counter > 6) {
        for (int i = 0; i < edges.size(); i++) {
            int set1 = find(subsets, edges[i].src);
            int set2 = find(subsets, edges[i].dest);

            if (set1 == set2) {
                continue;
            }

            if (cheapest[set1] == -1 || edges[cheapest[set1]].weight > edges[i].weight) {
                cheapest[set1] = i;
            }
            if (cheapest[set2] == -1 || edges[cheapest[set2]].weight > edges[i].weight) {
                cheapest[set2] = i;
            }
        }

        for (int v = 0; v < V; v++) {

            if (cheapest[v] == -1)  {
                continue;
            }

            int set1 = find(subsets, edges[cheapest[v]].src);
            int set2 = find(subsets, edges[cheapest[v]].dest);

            if (set1 != set2) {
                selected.push_back(edges[cheapest[v]]);
                cur_weight += edges[cheapest[v]].weight;
                UnionSets(subsets, set1, set2);
                tree_counter--;
            }

            cheapest[v] = -1;
        }
    }

    OutData(selected, cur_weight);
}

int main() {
    InitGraph();
    Algo_S(26);

    return 0;
}
```

c)

(c) Use algorithm S to produce a minimum spanning tree for the weighted graph given in task 3.

Output of my code :

```
MST Weight: 36
Selected Edges:
a -- d Weight: 1
b -- e Weight: 1
c -- e Weight: 2
c -- f Weight: 2
g -- k Weight: 2
h -- l Weight: 1
f -- i Weight: 2
i -- j Weight: 3
i -- m Weight: 2
n -- r Weight: 1
o -- p Weight: 2
p -- q Weight: 1
p -- s Weight: 2
p -- t Weight: 1
s -- z Weight: 2
a -- b Weight: 2
k -- r Weight: 2
e -- h Weight: 3
m -- p Weight: 2
n -- q Weight: 2
```

Task 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, $n = 1, 2, \ldots$ 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)

$\lim_{n\to\infty} \frac{2|E|}{|V||V-1|} = \lim_{n\to\infty} \frac{2n(n-1)}{2n(n-1)} = 1$. **Dense**

b)

(b) C_n (cycle graph)

$\lim_{n\to\infty} \frac{2|E|}{|V||V-1|} = \lim_{n\to\infty} \frac{2n}{n(n-1)} = \lim_{n\to\infty} \frac{2}{n-1} = 0$. **Sparse**

c)

(c) $K_{n,n}$ (complete bipartite)

$\lim_{n\to\infty} \frac{2|E|}{|V||V-1|} = \lim_{n\to\infty} \frac{2n^2}{2n(2n-1)} = \lim_{n\to\infty} \frac{n}{2n-1} = \frac{1}{2}$. **Dense**

d)

(d) $K_{3,n}$ (complete bipartite)

$\lim_{n\to\infty} \frac{2|E|}{|V||V-1|} = \lim_{n\to\infty} \frac{6n}{(n+3)(n+2)} = \lim_{n\to\infty} \frac{6n}{n^2+5n+6} = 0$. **Sparse**

e)

(e) Q_n (hypercube graph)

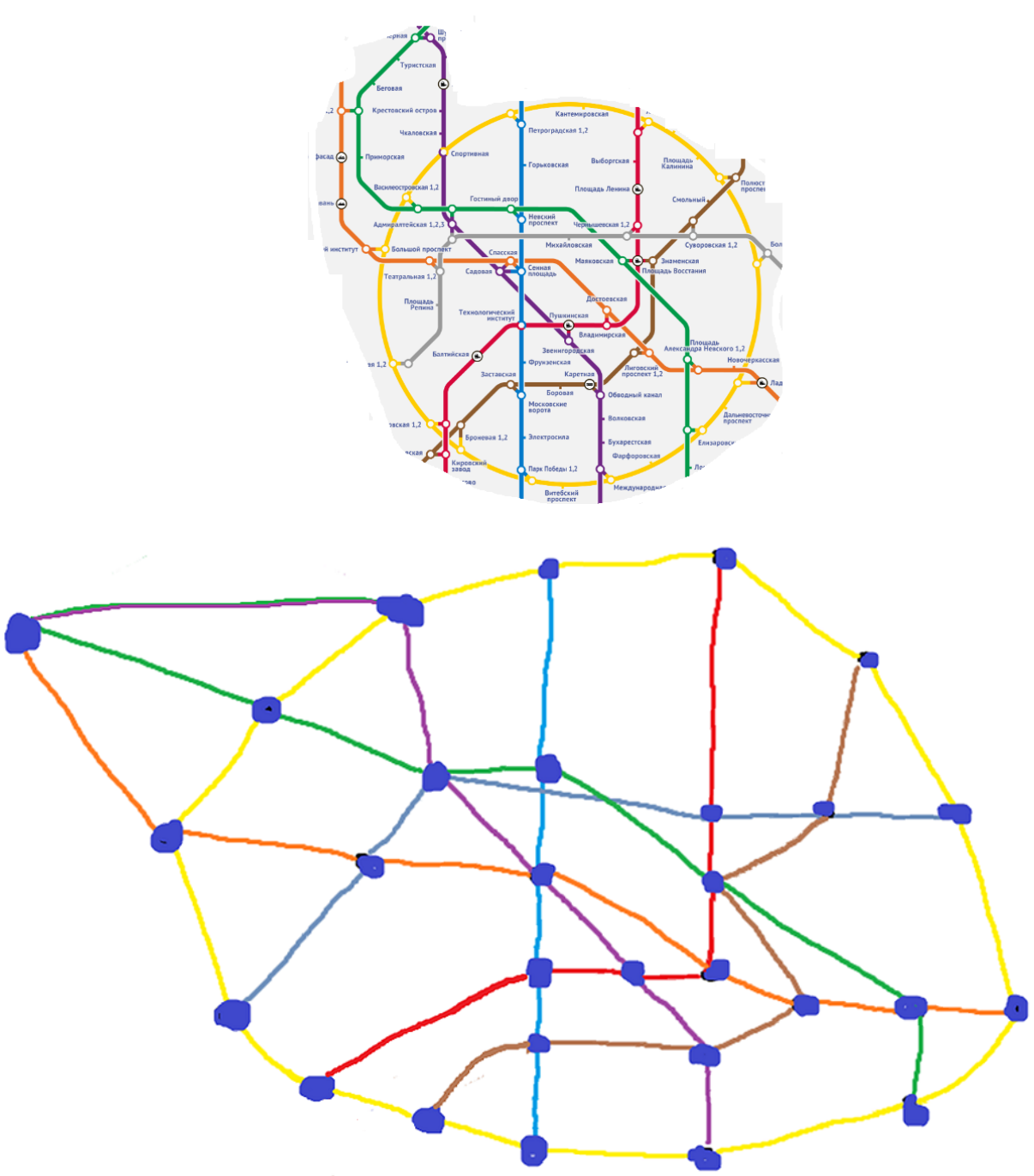
$\lim_{n\to\infty} \frac{2|E|}{|V||V-1|} = \lim_{n\to\infty} \frac{2^n n}{2^n(2^n-1)} = \lim_{n\to\infty} \frac{n}{2^n-1} = 0$. **Sparse**

f)

(f) W_n (wheel graph)

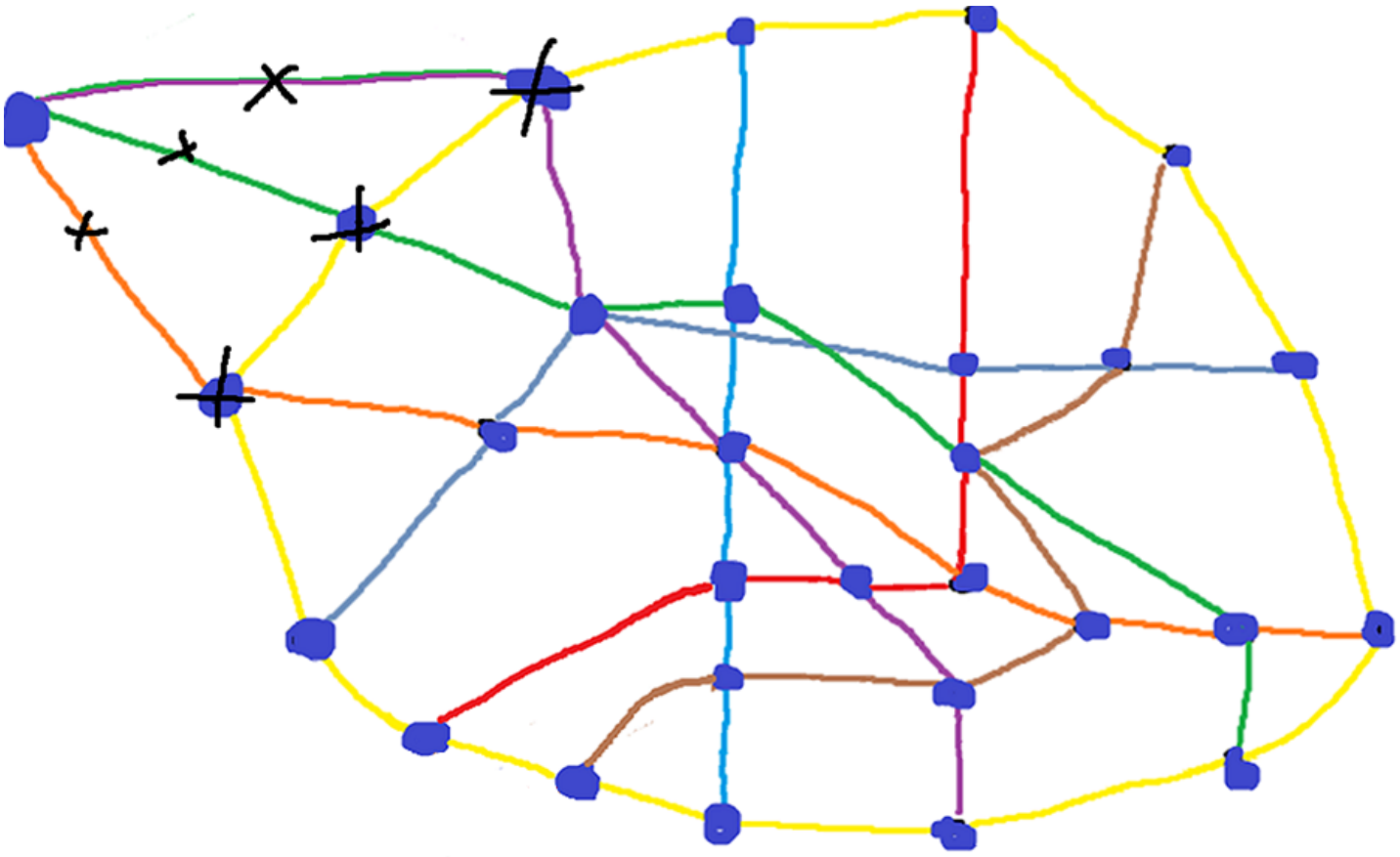
$\lim_{n\to\infty} \frac{2|E|}{|V||V-1|} = \lim_{n\to\infty} \frac{4(n-1)}{n(n-1)} = \lim_{n\to\infty} \frac{4}{n} = 0$. **Sprase**

Task 12

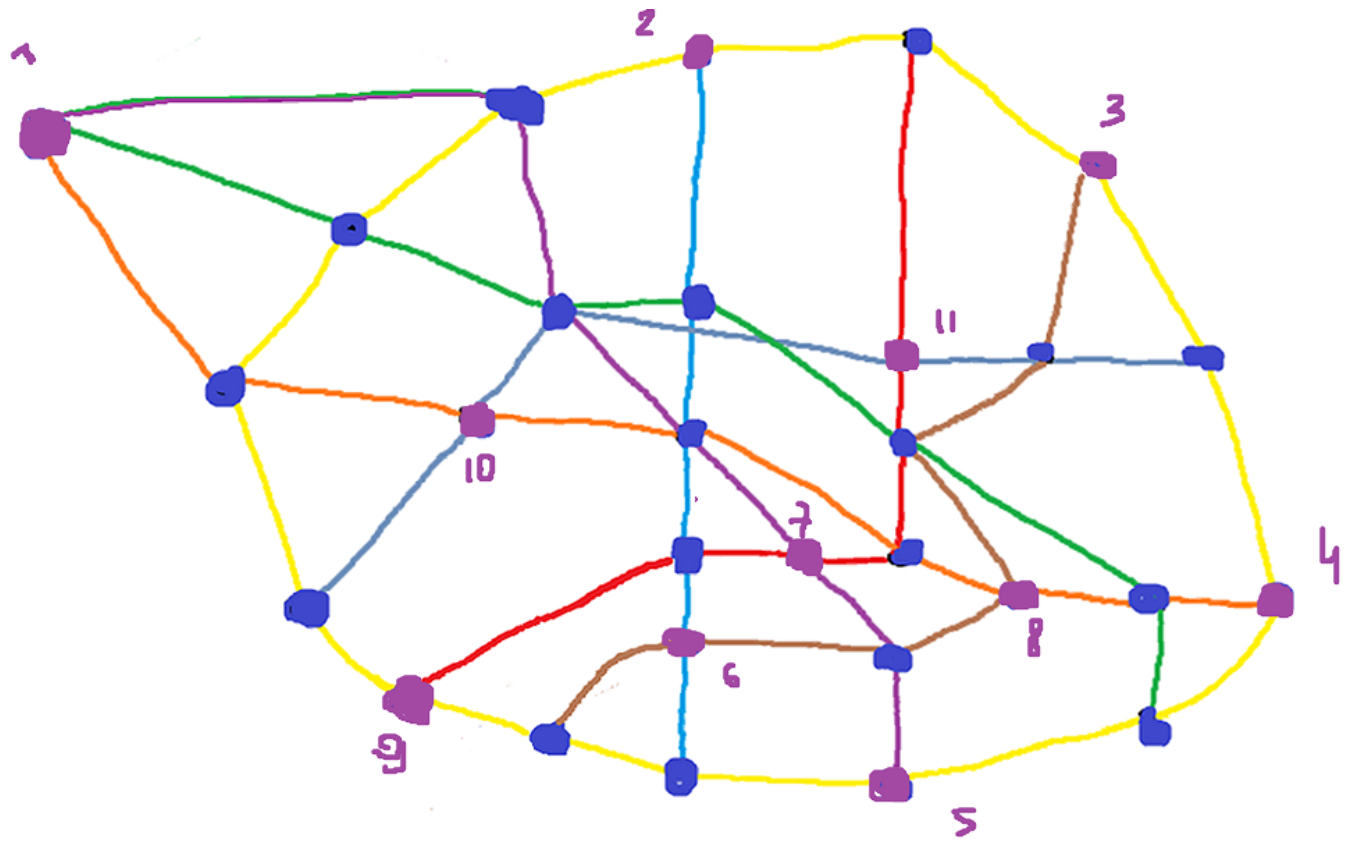


Vertex connectivity - 3

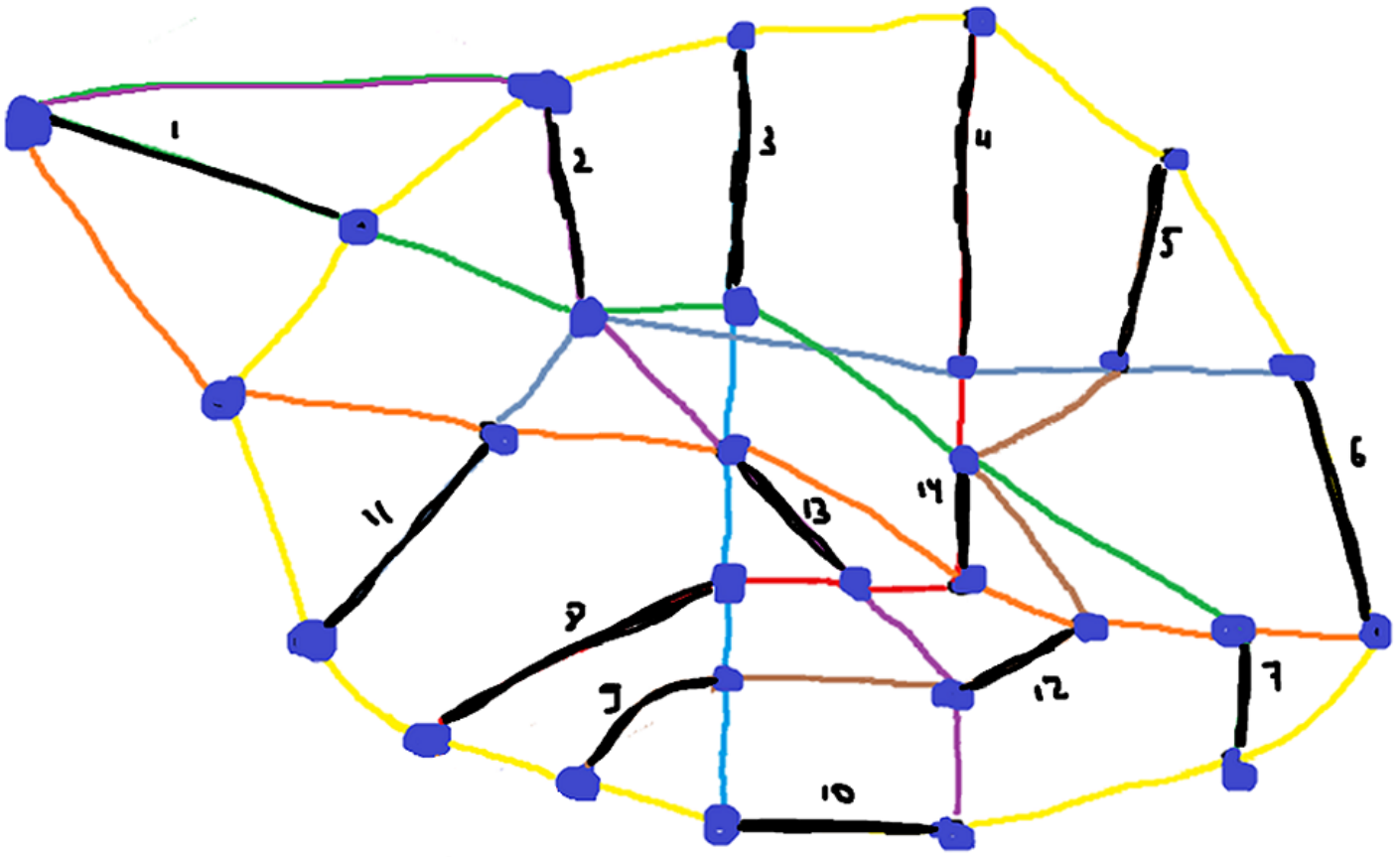
Edge connectivity - 3



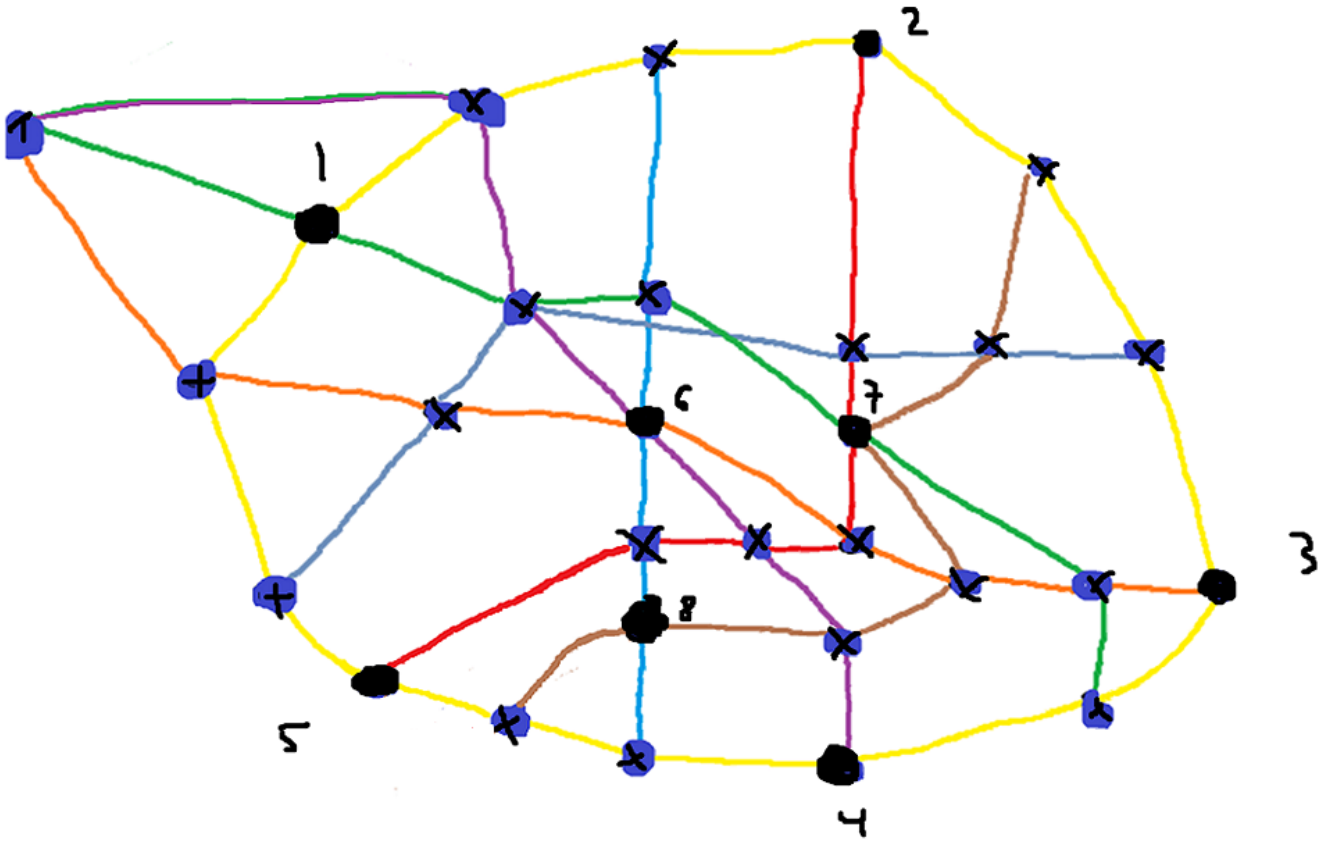
Maximum stable set : |V| = 11 (purple vertex)



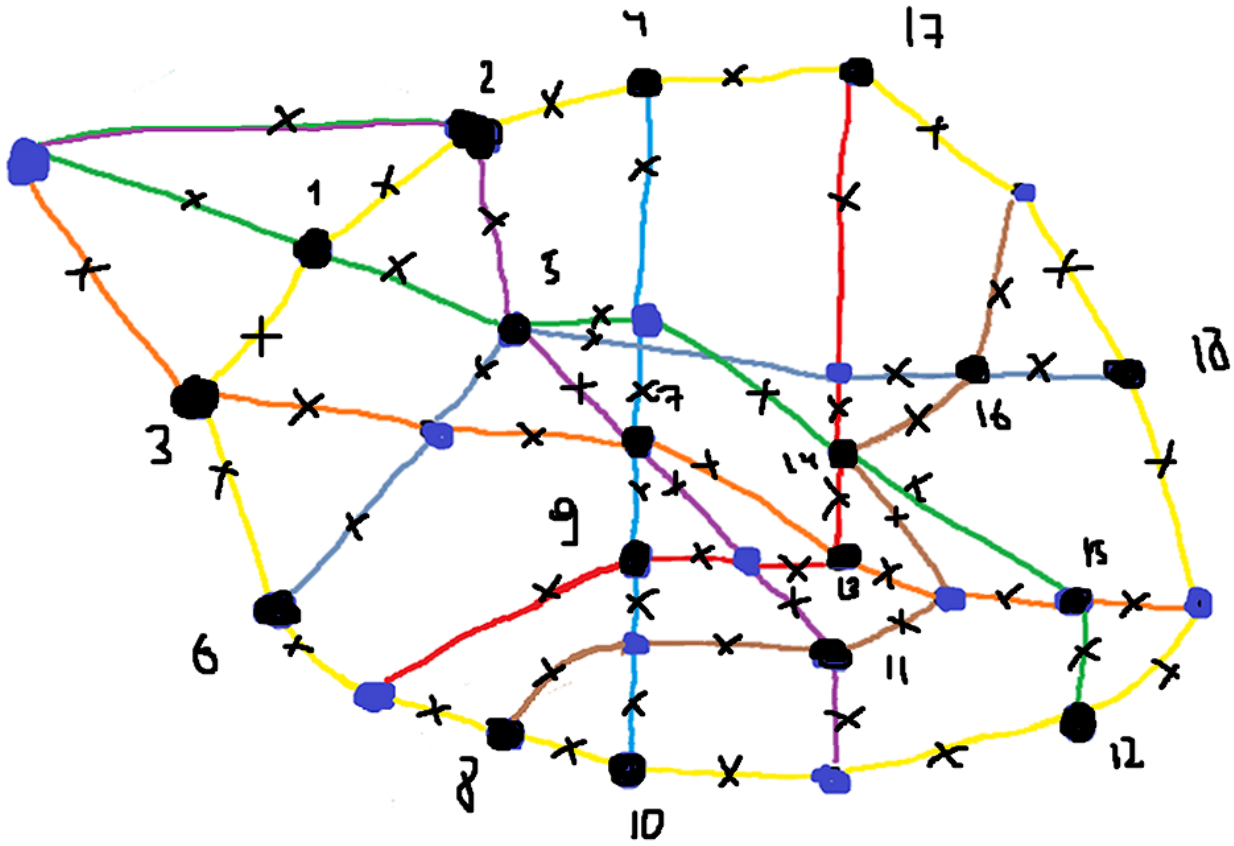
Maximum matching : |E| = 14(black edges)



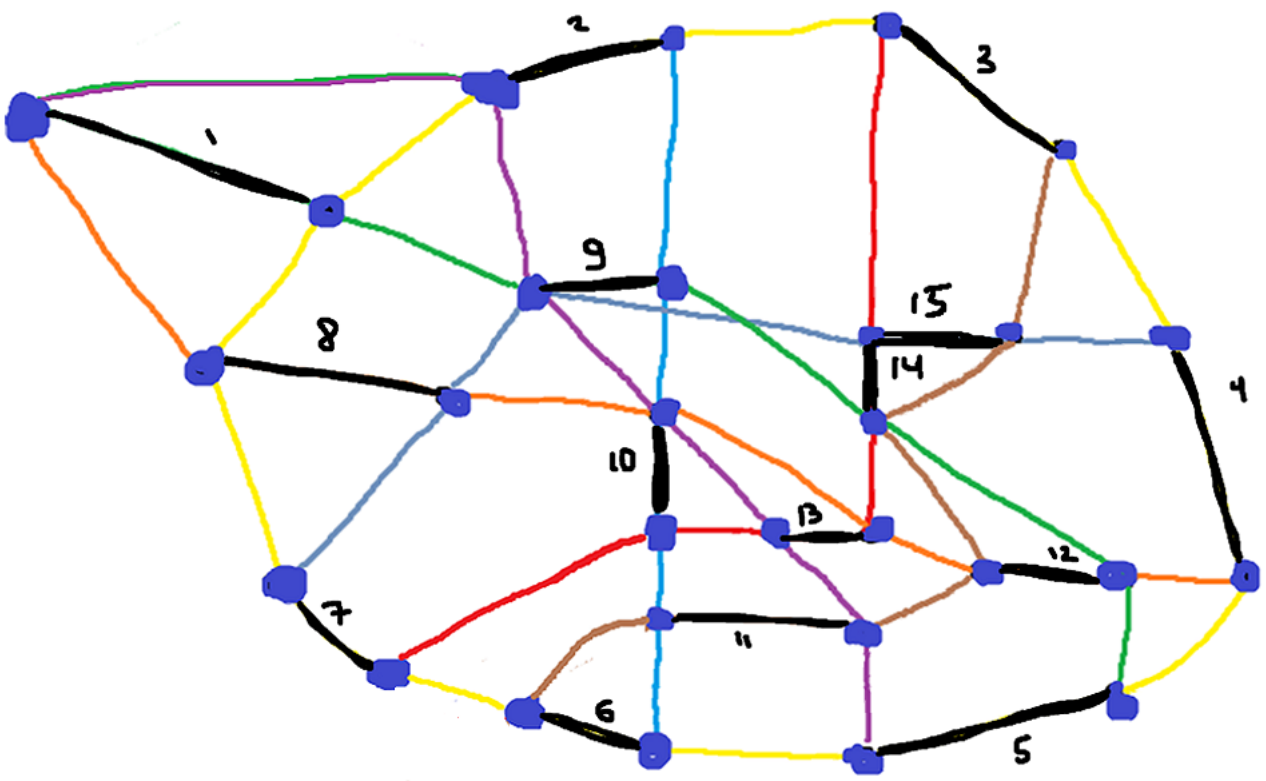
Minimum dominating set : |V| = 8(black vertex)



Minimum vertex cover : |V| = 18(black vertex)



Minimum edge cover : |E|= 15(black edges)



Task 13

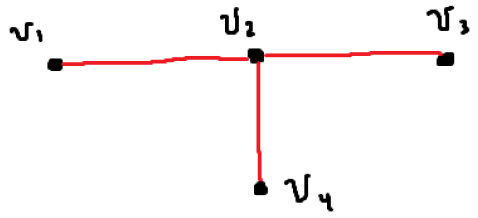
Find an error in the following inductive “proof” of the statement that every tree with n vertices has a path of length $n-1$.

▹ 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. □

Ошибка в приведенном индуктивном доказательстве заключается в утверждении, сделанном на индуктивном шаге.

Так как не каждое дерево, имеющее n вершин имеет $path$ длиной $n-1$.

Приведу пример дерева для которого это не выполняется :



Task 14

a)

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

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

💡 **Definition: distance between two vertices is the length of the shortest path from u to v.**

Предположим, что $dist(x,y) + dist(y,z) < dist(x,z)$. Тогда на самом деле у нас есть путь $x \to z$, который проходит через вершину y и имеет длину $dist(x,y) + dist(y,z)$. Но по определению $dist(x,z)$ это кратчайший путь, но у нас есть путь длина которого меньше, чем $dist(x,z)$. Противоречие

Таким образом:

$dist(x,y) + dist(y,z) \geq dist(x,z)$

b)

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

💡 $E(v) = max_{u \in V} dist(v,u)$

$rad(G) = min_{v \in V} E(v)$

$diam(G) = max_{v \in V} E(v)$
Так как диаметр это максимальный из
$E(v)$, а радиус это минимальный из $E(v)$. Значит, что $rad(G) \leq diam(G)$.
Возьмем
y как центральную вершину, значит $E(y) = rad(G)$ - это значит, что ни одна вершина не находится на расстоянии больше $rad(G)$ от вершины y. А также возьмем a и b, такие, что $dist(a,b) = diam(G)$. Тогда :

$dist(a,b) = diam(G) \leq dist(a,y) + dist(y,b) \leq rad(G) + rad(G) = 2rad(G)$

c)

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

💡 Докажем с помощью математической индукции.
База : дерево с 1 вершиной имеет 0 ребер.
Переход : дерево с
n вершинами имеет $n-1$ ребер.
Рассмотрим дерево с
$n+1$ вершинами. Мы всегда можем найти лист в этом дереве, потому что если в этом дереве нет листьев, то в этом дереве есть цикл, и это противоречие. Удалим этот лист и у нас останется n вершин и $n-1$ ребер.

Допустим у нас есть связанный граф $G(V,E)$ и $|E| = n-1, |V| = n$.
Если это не дерево, то там есть хотя бы 1 цикл. Предположим, что мы смогли удалить
m ребер так, что в графе больше нет циклов. Теперь это связанный ацикличный граф, то есть дерево. И в нем есть n вершин, а также $n-m-1$ ребер. Но это противоречит условию дерева $|E| = |V| - 1$.
Таким образом это не могло быть не деревом. Противоречие. Значит
$G(V,E) -$ ДЕРЕВО

d)

Theorem 4 (Whitney). For any graph G: $\varkappa(G) \leq \lambda(G) \leq \delta(G)$.

Докажем, что $k(G) \leq \lambda(G)$.
1) Граф G полный.

Тогда для того, чтобы сделать граф несвязанным или тривиальным нам надо удалить хотя бы $|V| - 1$ вершин или $|V| - 1$ ребер. Тогда $k(G) = \lambda(G) = |V| - 1$.

2) Граф G не полный.
1. G несвязанный или тривиальный, тогда
$k(G) = \lambda(G) = 0$
2. Граф имеет мост, тогда
$k(G) = \lambda(G) = 1$. Так как мы можем удалить мост или вершину, инцидентную ему.
3. Если
$\lambda(G) \geq 2$.
В этом случае мы хотим соотнести удаленные ребра с удаленными вершинами. Допустим мы начали удалять ребра один за другим до тех пор, пока новый подграф
G' графа G не будет обладать свойством $\lambda(G') = 1$. То есть обладать мостом, который связывает вершины a и b.
Теперь начнем с исходного графа
G. Мы снова начинаем сокращать наш граф, но теперь мы удаляем вершины, которые инцидентны нашим удаленным ребрам и отличаются от a и b. Мы делаем это до тех пор, пока не поймем, что нет вершин, отличных от a и b. Если к этому моменту граф уже стал несвязным, то
$k(G) \leq \lambda(G)$. В противном случае мы можем удалить a или b и граф станет несвязным или тривиальным, то есть
$k(G) = \lambda(G)$.

Таким образом $k(G) \leq \lambda(G)$.

Теперь докажем, что
$\lambda(G) \leq \delta(G)$.

Если $|E(G)| = 0$, то $\lambda(G) = \delta(G) = 0$.
Иначе найдем вершину
v, такую что $deg(v) = \delta(G)$ и удалим все ее ребра. Граф G стал несвязным, поэтому $\lambda(G)$ должно быть меньше или равно $deg(v)$, то есть $\lambda(G) \leq \delta(G)$.

e)

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

Заметим, что если $\delta(G) \geq ||V|/2|$, то $diam(G) \leq 2$.
Если
$diam(G) = 1$, то граф является полным и $\lambda(G) = \delta(G) = |V| - 1$.
Иначе
$diam(G) = 2$. Рассмотрим множество T, такое, что $|T| = \lambda(G)$ и T будет содержать все ребра, при удалении которых исходный граф становится несвязным. Пусть A и B это компоненты, которые сформировались после удаления ребер. Если есть вершина из A, которая не инцидентна какому-то ребру из T, а также есть вершина из B, которая не инцидентна некоторому ребру из T, то $diam(G) > 2$. Это значит, что либо A или B имеет вершину, которая инцидентна некоторому ребру из T.
Допустим, что вершина
$v \in A$ инцидентна некоторому ребру из T. Возьмем x как количество ребер, инцидентных v. Тогда по крайней мере $deg(v) - x$ соседей v в A. Другие вершины из A инцидентны как минимум 1 из $|T| - x$ ребер, не инцидентных v.
Таким образом верно следующее:
1)
$\lambda(G) - x = |T| - x$
2)
$|T| - x \geq |A \backslash v|$(так как каждая вершина $u \neq v$ из A инцидентна как минимум 1 из $|T| - x$ ребер не инцидентных v
.
3)
$|A \backslash v| \geq deg(v) - x$(так как количество вершин $u \neq v$ в A больше количества соседей v.
4)
$deg(v) - x \geq \delta(G) - x$(так как $\delta(G) = min\, deg(u) \forall u \in G$
Тогда
$\lambda(G) - x \geq \delta(G) - x$, то есть $\lambda(G) \geq \delta(G)$.
Но по теореме 4(Whitney)
$\lambda(G) \leq \delta(G)$. Следовательно $\lambda(G) = \delta(G)$.

f)

Theorem 6 (Harary). Every block of a block graph[1] is a clique.

Предположим, в блочном графе B имеется блок не являющийся кликой. Тогда в этом блоке имеется по крайней мере 2 вершины, которые не соединены ребром, то есть они лежат на общем цикле длиной больше или равной 4. Объединение блоков, которые не связаны с этими вершинами, связано и эти блоки не имеют общей вершины разреза, поэтому они должны содержаться в блоке большего размера. Но так как блок это максимальный двусвязный подграф, то мы получаем противоречие, так как можем увеличить маленькие блоки до блока большего размера.
То есть все блоки в блочном графе это клики.