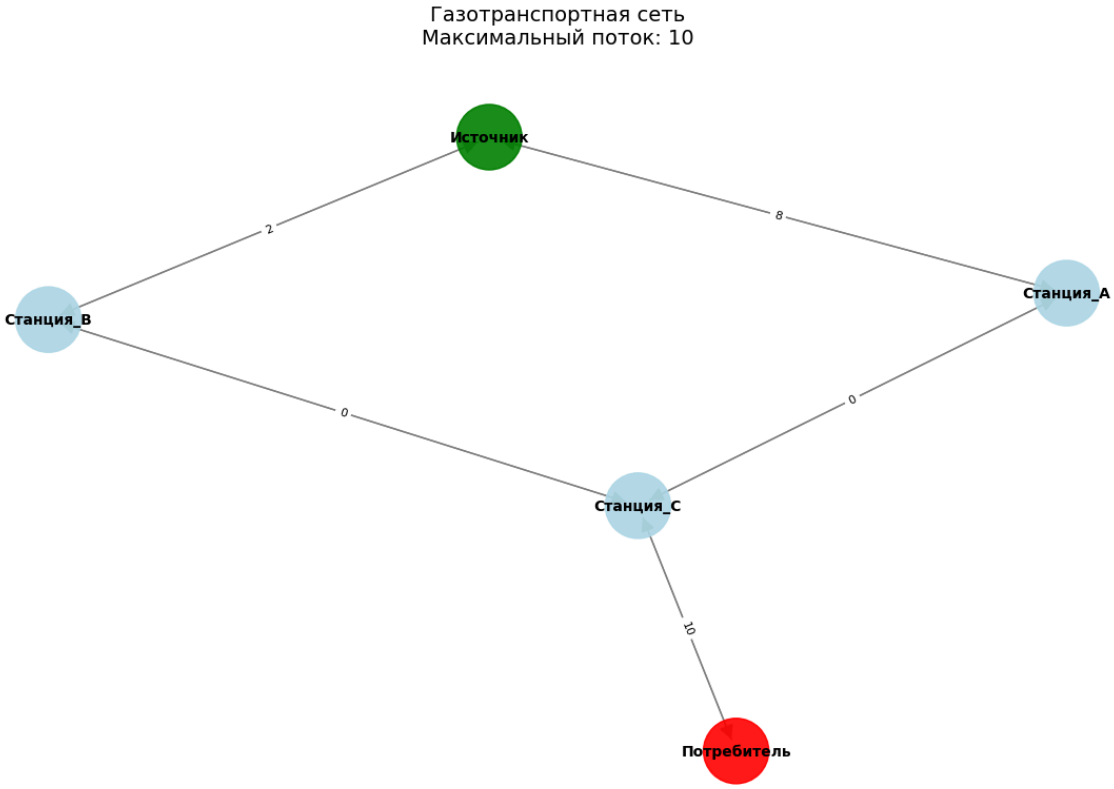
задача 1

```
from collections import defaultdict, deque
import matplotlib.pyplot as plt
import networkx as nx

class GasNetworkFlow:
    def __init__(self):
        # Граф в виде словаря: graph[источник][назначение] = пропускная способность
        self.graph = defaultdict(lambda: defaultdict(int))
        # Хранилище фактических потоков
        self.flow = defaultdict(lambda: defaultdict(int))
        # Множество всех узлов сети
        self.nodes = set()

    def add_edge(self, source: str, sink: str, capacity: int):
        """Добавляет ребро в граф"""
        self.graph[source][sink] += capacity
        # Инициализируем обратное ребро
        if sink not in self.graph or source not in self.graph[sink]:
            self.graph[sink][source] = 0
        self.nodes.add(source)
        self.nodes.add(sink)

    def bfs(self, source: str, sink: str, parent: dict) -> bool:
        """Поиск в ширину для нахождения пути"""
        visited = set()
        queue = deque([source])
        visited.add(source)

        while queue:
            u = queue.popleft()

            for v in self.graph[u]:
                if v not in visited and self.graph[u][v] > 0:
                    visited.add(v)
                    queue.append(v)
                    parent[v] = u
                    if v == sink:
                        return True
        return False

    def ford_fulkerson(self, source: str, sink: str) -> int:
        """Алгоритм Форда-Фалкерсона для нахождения максимального потока"""
        max_flow = 0
        parent = {}

        # Пока есть увеличивающие пути
        while self.bfs(source, sink, parent):
            # Находим минимальную пропускную способность на пути
            path_flow = float('inf')
            s = sink
            while s != source:
                u = parent[s]
                path_flow = min(path_flow, self.graph[u][s])
                s = u

            # Увеличиваем общий поток
            max_flow += path_flow

            # Обновляем остаточные пропускные способности
            v = sink
            while v != source:
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                self.flow[u][v] += path_flow
                v = u

            parent = {}

        return max_flow

    def visualize_network(self, source: str, sink: str, max_flow: int):
        """Визуализация графа сети"""
        G = nx.DiGraph()

        # Добавляем узлы
        for node in self.nodes:
            G.add_node(node)
```

```
        # Добавляем рёбра
        edge_labels = {}
        for u in self.nodes:
            for v in self.nodes:
                if self.graph[u][v] > 0 or self.flow[u][v] > 0:
                    G.add_edge(u, v)
                    edge_labels[(u, v)] = f"{self.flow[u][v]}"

        # Создаем расположение узлов
        pos = nx.spring_layout(G, seed=42)

        # Рисуем граф
        plt.figure(figsize=(12, 8))

        # Рисуем узлы
        node_colors = []
        for node in G.nodes():
            if node == source:
                node_colors.append('green')
            elif node == sink:
                node_colors.append('red')
            else:
                node_colors.append('lightblue')

        nx.draw_networkx_nodes(G, pos, node_color=node_colors,
                               node_size=2000, alpha=0.9)

        # Рисуем рёбра
        nx.draw_networkx_edges(G, pos, edge_color='gray',
                               arrows=True, arrowsize=20)

        # Подписи узлов и рёбер
        nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold')
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
                                     font_size=8)

        plt.title(f"Газотранспортная сеть\nМаксимальный поток: {max_flow}",
                  fontsize=14, pad=20)
        plt.axis('off')
        plt.tight_layout()
        plt.show()

def main():
    """Основная функция демонстрации"""
    print("="*50)
    print("ГАЗОТРАНСПОРТНАЯ СЕТЬ - ДЕМОНСТРАЦИЯ")
    print("="*50)

    # Создаем сеть
    network = GasNetworkFlow()

    # Простая тестовая сеть
    network.add_edge("Источник", "Станция_A", 10)
    network.add_edge("Источник", "Станция_B", 5)
    network.add_edge("Станция_A", "Станция_C", 8)
    network.add_edge("Станция_B", "Станция_C", 3)
    network.add_edge("Станция_C", "Потребитель", 10)

    print("Сеть создана:")
    print("  Источник → Станция_A: 10")
    print("  Источник → Станция_B: 5")
    print("  Станция_A → Станция_C: 8")
    print("  Станция_B → Станция_C: 3")
    print("  Станция_C → Потребитель: 10")

    # Вычисляем максимальный поток
    source = "Источник"
    sink = "Потребитель"
    max_flow = network.ford_fulkerson(source, sink)

    print(f"\nРезультат:")
    print(f"Максимальный поток: {max_flow}")

    # Показываем потоки по рёбрам
    print("\nПотоки по рёбрам:")
    for u in network.nodes:
        for v in network.nodes:
            if network.flow[u][v] > 0:
                print(f"  {u} → {v}: {network.flow[u][v]}")

    # Визуализируем
```

```
    network.visualize_network(source, sink, max_flow)

if __name__ == "__main__":
    main()
```

```
==================================================
ГАЗОТРАНСПОРТНАЯ СЕТЬ - ДЕМОНСТРАЦИЯ
==================================================
Сеть создана:
  Источник → Станция_A: 10
  Источник → Станция_B: 5
  Станция_A → Станция_C: 8
  Станция_B → Станция_C: 3
  Станция_C → Потребитель: 10

Результат:
Максимальный поток: 10

Потоки по рёбрам:
  Станция_B → Станция_C: 2
  Станция_A → Станция_C: 8
  Источник → Станция_B: 2
  Источник → Станция_A: 8
  Станция_C → Потребитель: 10
```



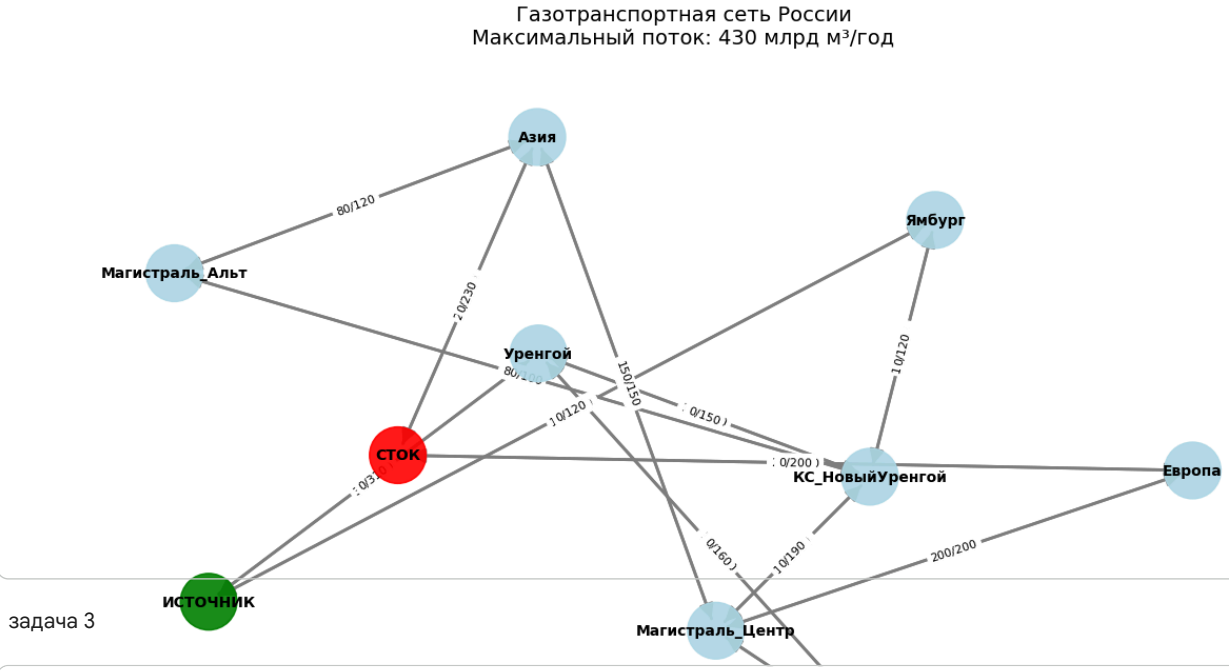
Задача 2

```
from collections import defaultdict, deque
import matplotlib.pyplot as plt
import networkx as nx

class GasTransportNetwork:
    def __init__(self):
        # Граф: источник -> назначение -> пропускная способность
        self.graph = defaultdict(lambda: defaultdict(int))
        # Фактические потоки по рёбрам
        self.flow = defaultdict(lambda: defaultdict(int))
        # Все узлы сети
        self.nodes = set()

    def add_edge(self, source: str, sink: str, capacity: int):
        """Добавляет газопровод в сеть"""
        self.graph[source][sink] = capacity
        # Обратное ребро для остаточного графа
        self.graph[sink][source] = 0
        self.nodes.add(source)
        self.nodes.add(sink)
```

```
    def bfs(self, source: str, sink: str, parent: dict) -> bool:
        """Поиск пути от источника к стоку"""
        visited = set([source])
        queue = deque([source])

        while queue:
            u = queue.popleft()
            for v in self.graph[u]:
                if v not in visited and self.graph[u][v] > 0:
                    visited.add(v)
                    queue.append(v)
                    parent[v] = u
                    if v == sink:
                        return True
        return False

    def ford_fulkerson(self, source: str, sink: str) -> int:
        """Вычисляет максимальный поток алгоритмом Форда-Фалкерсона"""
        max_flow = 0
        parent = {}

        # Пока есть увеличивающие пути
        while self.bfs(source, sink, parent):
            # Находим минимальную пропускную способность на пути
            path_flow = float('inf')
            s = sink
            while s != source:
                u = parent[s]
                path_flow = min(path_flow, self.graph[u][s])
                s = u

            # Обновляем поток и остаточные способности
            max_flow += path_flow
            v = sink
            while v != source:
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                self.flow[u][v] += path_flow
                v = u

            parent = {}

        return max_flow

    def visualize_network(self, source: str, sink: str, max_flow: int):
        """Строит визуализацию газотранспортной сети"""
        G = nx.DiGraph()

        # Добавляем узлы
        for node in self.nodes:
            G.add_node(node)

        # Добавляем рёбра с потоками
        edge_labels = {}
        for u in self.nodes:
            for v in self.nodes:
                if self.graph[u][v] > 0 or self.flow[u][v] > 0:
                    G.add_edge(u, v)
                    capacity = self.graph[u][v] + self.flow[u][v]  # Восстанавливаем исходную capacity
                    edge_labels[(u, v)] = f"{self.flow[u][v]}/{capacity}"

        # Располагаем узлы
        pos = nx.spring_layout(G, seed=42, k=2)

        # Рисуем граф
        plt.figure(figsize=(12, 8))

        # Цвета узлов
        node_colors = ['green' if node == source else
                       'red' if node == sink else
                       'lightblue' for node in G.nodes()]

        nx.draw_networkx_nodes(G, pos, node_color=node_colors,
                               node_size=1500, alpha=0.9)
        nx.draw_networkx_edges(G, pos, edge_color='gray',
                               arrows=True, arrowsize=20, width=2)
        nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold')
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
                                     font_size=8)

        plt.title(f"Газотранспортная сеть России\nМаксимальный поток: {max_flow} млрд м³/год",
```

```
                  fontsize=14, pad=20)
        plt.axis('off')
        plt.tight_layout()
        plt.show()

def main():
    """Демонстрация работы алгоритма"""
    print("="*60)
    print("АНАЛИЗ ГАЗОТРАНСПОРТНОЙ СЕТИ РОССИИ")
    print("="*60)

    # Создаём сеть
    network = GasTransportNetwork()

    print("\n1. СОЗДАНИЕ СЕТИ:")
    print("-"*40)

    # Упрощённая модель газотранспортной системы
    network.add_edge("ИСТОЧНИК", "Уренгой", 330)
    network.add_edge("ИСТОЧНИК", "Ямбург", 120)
    network.add_edge("Уренгой", "КС_Муравленко", 180)
    network.add_edge("Уренгой", "КС_НовыйУренгой", 150)
    network.add_edge("Ямбург", "КС_НовыйУренгой", 120)
    network.add_edge("КС_Муравленко", "Магистраль_Центр", 160)
    network.add_edge("КС_НовыйУренгой", "Магистраль_Центр", 190)
    network.add_edge("КС_НовыйУренгой", "Магистраль_Альт", 100)
    network.add_edge("Магистраль_Центр", "Европа", 200)
    network.add_edge("Магистраль_Центр", "Азия", 150)
    network.add_edge("Магистраль_Альт", "Азия", 120)
    network.add_edge("Европа", "СТОК", 200)
    network.add_edge("Азия", "СТОК", 270)

    # Выводим добавленные рёбра
    edges = [
        ("ИСТОЧНИК", "Уренгой", 330), ("ИСТОЧНИК", "Ямбург", 120),
        ("Уренгой", "КС_Муравленко", 180), ("Уренгой", "КС_НовыйУренгой", 150),
        ("Ямбург", "КС_НовыйУренгой", 120), ("КС_Муравленко", "Магистраль_Центр", 160),
        ("КС_НовыйУренгой", "Магистраль_Центр", 190), ("КС_НовыйУренгой", "Магистраль_Альт", 100),
        ("Магистраль_Центр", "Европа", 200), ("Магистраль_Центр", "Азия", 150),
        ("Магистраль_Альт", "Азия", 120), ("Европа", "СТОК", 200),
        ("Азия", "СТОК", 270)
    ]

    for u, v, cap in edges:
        print(f"  {u:20} → {v:20} : {cap:3} млрд м³/год")

    # Вычисляем максимальный поток
    print("\n2. ВЫЧИСЛЕНИЕ МАКСИМАЛЬНОГО ПОТОКА:")
    print("-"*40)

    max_flow = network.ford_fulkerson("ИСТОЧНИК", "СТОК")
    print(f"Максимальный поток: {max_flow} млрд м³/год")

    # Анализ результатов
    print("\n3. АНАЛИЗ РЕЗУЛЬТАТОВ:")
    print("-"*40)

    flow_europe = network.flow["Европа"]["СТОК"]
    flow_asia = network.flow["Азия"]["СТОК"]

    print(f"Поток в Европу: {flow_europe} млрд м³/год")
    print(f"Поток в Азию: {flow_asia} млрд м³/год")
    print(f"Общий экспорт: {flow_europe + flow_asia} млрд м³/год")

    # Визуализация
    print("\n4. ВИЗУАЛИЗАЦИЯ СЕТИ:")
    print("-"*40)
    network.visualize_network("ИСТОЧНИК", "СТОК", max_flow)

if __name__ == "__main__":
    main()
```

```
============================================================
АНАЛИЗ ГАЗОТРАНСПОРТНОЙ СЕТИ РОССИИ
============================================================

1. СОЗДАНИЕ СЕТИ:
----------------------------------------
  ИСТОЧНИК             → Уренгой             : 330 млрд м³/год
  ИСТОЧНИК             → Ямбург              : 120 млрд м³/год
  Уренгой              → КС_Муравленко       : 180 млрд м³/год
  Уренгой              → КС_НовыйУренгой     : 150 млрд м³/год
  Ямбург               → КС_НовыйУренгой     : 120 млрд м³/год
  КС_Муравленко        → Магистраль_Центр    : 160 млрд м³/год
  КС_НовыйУренгой      → Магистраль_Центр    : 190 млрд м³/год
  КС_НовыйУренгой      → Магистраль_Альт     : 100 млрд м³/год
  Магистраль_Центр     → Европа              : 200 млрд м³/год
  Магистраль_Центр     → Азия                : 150 млрд м³/год
  Магистраль_Альт      → Азия                : 120 млрд м³/год
  Европа               → СТОК                : 200 млрд м³/год
  Азия                 → СТОК                : 270 млрд м³/год

2. ВЫЧИСЛЕНИЕ МАКСИМАЛЬНОГО ПОТОКА:
----------------------------------------
Максимальный поток: 430 млрд м³/год

3. АНАЛИЗ РЕЗУЛЬТАТОВ:
----------------------------------------
Поток в Европу: 200 млрд м³/год
Поток в Азию: 230 млрд м³/год
Общий экспорт: 430 млрд м³/год

4. ВИЗУАЛИЗАЦИЯ СЕТИ:
----------------------------------------
```



задача 3

```
import plotly.graph_objects as go
import networkx as nx
from collections import defaultdict, deque
from typing import Dict, Set, List, Tuple
import matplotlib.pyplot as plt

class GasTransportNetwork:
    def __init__(self):
        # Инициализация графа как словаря словарей для хранения пропускных способностей
        self.graph = defaultdict(lambda: defaultdict(int))
        # Словарь для хранения фактических потоков
        self.flow = defaultdict(lambda: defaultdict(int))
        # Множество всех узлов сети
        self.nodes = set()
        # Словарь для хранения исходных пропускных способностей
        self.original_capacities = defaultdict(lambda: defaultdict(int))
        # Словарь для ограничений пропускной способности вершин
        self.node_capacity = defaultdict(lambda: float('inf'))

    def set_node_capacity(self, node: str, capacity: int):
        """Устанавливает максимальную пропускную способность вершины."""
        self.node_capacity[node] = capacity

    def add_edge(self, source: str, sink: str, capacity: int):
        """
        Добавляет ребро в граф с указанной пропускной способностью.
```

```
        Args:
            source: исходный узел
            sink: целевой узел
            capacity: пропускная способность
        """
        self.graph[source][sink] += capacity
        self.original_capacities[source][sink] += capacity
        # Инициализируем обратное ребро нулевой пропускной способностью
        if sink not in self.graph or source not in self.graph[sink]:
            self.graph[sink][source] = 0
            self.original_capacities[sink][source] = 0
        # Добавляем узлы в множество
        self.nodes.add(source)
        self.nodes.add(sink)

    def add_edge_with_node_constraints(self, source: str, sink: str, capacity: int):
        """
        Добавляет ребро, учитывая ограничения на вершины.
        Решение: разбить вершину на две с ограничивающим ребром.
        """
        # Разбиваем вершину sink на входную и выходную части
        sink_in = f"{sink}_in"
        sink_out = f"{sink}_out"

        # Добавляем основное ребро от source к входной части sink
        self.add_edge(source, sink_in, capacity)

        # Добавляем ограничивающее ребро между входной и выходной частями
        self.add_edge(sink_in, sink_out, self.node_capacity[sink])

        # Возвращаем имена созданных узлов для дальнейшего использования
        return sink_in, sink_out

    def bfs(self, source: str, sink: str, parent: Dict) -> bool:
        """
        Поиск в ширину для нахождения увеличивающего пути.

        Returns:
            True если путь от source до sink существует, иначе False
        """
        visited = set()
        queue = deque([source])
        visited.add(source)

        while queue:
            u = queue.popleft()

            # Проверяем всех соседей текущего узла
            for v in self.graph[u]:
                # Если узел не посещен и есть остаточная пропускная способность
                if v not in visited and self.graph[u][v] > 0:
                    visited.add(v)
                    queue.append(v)
                    parent[v] = u
                    # Если достигли сток, путь найден
                    if v == sink:
                        return True
        return False

    def ford_fulkerson(self, source: str, sink: str) -> int:
        """
        Реализация алгоритма Форда-Фалкерсона для нахождения максимального потока.

        Returns:
            Значение максимального потока
        """
        # Создаем рабочую копию графа для алгоритма
        self.working_graph = defaultdict(lambda: defaultdict(int))
        for u in self.graph:
            for v in self.graph[u]:
                self.working_graph[u][v] = self.graph[u][v]

        max_flow_value = 0
        parent = {}

        # Пока существуют увеличивающие пути
        while self.bfs(source, sink, parent):
            # Находим минимальную пропускную способность на пути
            path_flow = float('inf')
            s = sink
```

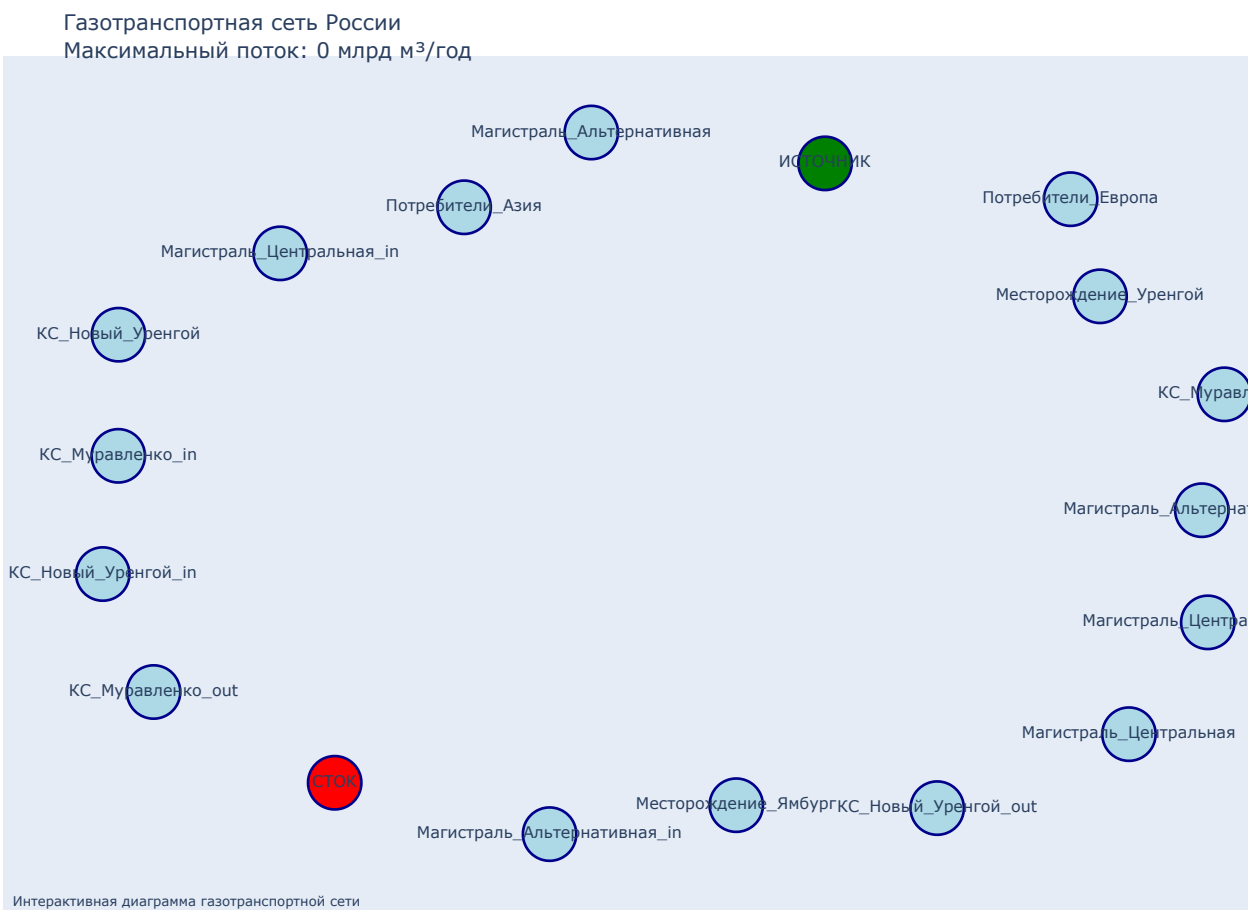
```
            # Проходим путь от стока к источнику для нахождения узкого места
            while s != source:
                u = parent[s]
                path_flow = min(path_flow, self.working_graph[u][s])
                s = u

            # Увеличиваем общий поток
            max_flow_value += path_flow

            # Обновляем остаточные пропускные способности
            v = sink
            while v != source:
                u = parent[v]
                self.working_graph[u][v] -= path_flow
                self.working_graph[v][u] += path_flow
                self.flow[u][v] += path_flow
                v = u

            # Сбрасываем родительский словарь для следующей итерации
            parent = {}

        return max_flow_value

    def get_flow_on_edge(self, u: str, v: str) -> int:
        """Возвращает поток по конкретному ребру."""
        return self.flow[u][v]

    def create_interactive_visualization(self, source: str, sink: str, max_flow_value: int):
        """
        Создает интерактивную визуализацию графа с использованием Plotly.
        """
        # Создаем ориентированный граф для визуализации
        G = nx.DiGraph()

        # Добавляем узлы
        for node in self.nodes:
            G.add_node(node)

        # Добавляем рёбра и собираем данные для визуализации
        edge_traces = []
        edge_labels = []
        edge_x = []
        edge_y = []

        # Создаем позиции узлов с использованием spring layout
        pos = nx.spring_layout(G, k=3, iterations=50)

        # Создаем трассировку для рёбер
        for u, v in G.edges():
            if self.original_capacities[u][v] > 0:
                x0, y0 = pos[u]
                x1, y1 = pos[v]

                # Определяем цвет и ширину ребра в зависимости от загрузки
                capacity = self.original_capacities[u][v]
                flow = self.flow[u][v]

                if capacity > 0:
                    utilization = flow / capacity
                    if utilization == 1:
                        color = 'red'
                        width = 6
                    elif utilization > 0.7:
                        color = 'orange'
                        width = 4
                    elif utilization > 0:
                        color = 'blue'
                        width = 3
                    else:
                        color = 'gray'
                        width = 1
                else:
                    color = 'gray'
                    width = 1

                # Создаем трассировку для ребра
                edge_trace = go.Scatter(
                    x=[x0, x1, None], y=[y0, y1, None],
                    line=dict(width=width, color=color),
                    hoverinfo='text',
                    mode='lines')
```

```
            edge_traces.append(edge_trace)

            # Добавляем подпись для ребра
            edge_labels.append(
                go.Scatter(
                    x=[(x0+x1)/2], y=[(y0+y1)/2],
                    text=[f"{u}→{v}<br>{flow}/{capacity}"],
                    mode='text',
                    textposition="middle center",
                    hoverinfo='none',
                    textfont=dict(size=10, color='black')
                )
            )

    # Создаем трассировку для узлов
    node_x = []
    node_y = []
    node_text = []
    node_color = []

    for node in G.nodes():
        x, y = pos[node]
        node_x.append(x)
        node_y.append(y)
        node_text.append(node)

        # Определяем цвет узла
        if node == source:
            node_color.append('green')
        elif node == sink:
            node_color.append('red')
        else:
            node_color.append('lightblue')

    node_trace = go.Scatter(
        x=node_x, y=node_y,
        mode='markers+text',
        hoverinfo='text',
        text=node_text,
        textposition="middle center",
        marker=dict(
            color=node_color,
            size=40,
            line=dict(width=2, color='darkblue')
        )
    )

    # Создаем фигуру
    fig = go.Figure()

    # Добавляем все трассировки
    for trace in edge_traces:
        fig.add_trace(trace)

    for label in edge_labels:
        fig.add_trace(label)

    fig.add_trace(node_trace)

    # Настраиваем layout
    fig.update_layout(
        title=f'Газотранспортная сеть России<br>Максимальный поток: {max_flow_value} млрд м³/год',
        titlefont_size=16,
        showlegend=False,
        hovermode='closest',
        margin=dict(b=20,l=5,r=5,t=40),
        annotations=[ dict(
            text="Интерактивная диаграмма газотранспортной сети",
            showarrow=False,
            xref="paper", yref="paper",
            x=0.005, y=-0.002,
            xanchor='left', yanchor='bottom',
            font=dict(size=10)
        ) ],
        xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
        yaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
        width=1000,
        height=700
    )

    return fig
```

```
class CostFlowNetwork(GasTransportNetwork):
    """
    Расширение для учёта стоимости транспортировки газа:
    не только максимальный поток, но и минимальная стоимость.
    """
    def __init__(self):
        super().__init__()
        # Словарь для хранения стоимостей транспортировки
        self.cost = defaultdict(lambda: defaultdict(int))
        # Словарь для хранения потоков с учетом стоимости
        self.flow_with_cost = defaultdict(lambda: defaultdict(int))

    def add_edge_with_cost(self, u: str, v: str, capacity: int, cost: float):
        """
        Добавляет ребро с учётом стоимости транспортировки.

        Args:
            cost: стоимость транспортировки единицы потока (руб./млрд м³)
        """
        # Добавляем ребро в граф пропускных способностей
        self.graph[u][v] += capacity
        self.original_capacities[u][v] += capacity
        # Инициализируем обратное ребро
        self.graph[v][u] = 0
        self.original_capacities[v][u] = 0
        # Сохраняем стоимость транспортировки
        self.cost[u][v] = cost
        self.cost[v][u] = -cost  # Отрицательная стоимость для обратного ребра
        # Добавляем узлы в сет
        self.nodes.add(u)
        self.nodes.add(v)

    def min_cost_max_flow(self, source: str, sink: str) -> Tuple[int, float]:
        """
        Находит максимальный поток минимальной стоимости.

        Returns:
            Кортеж (максимальный поток, общая стоимость)
        """
        # Инициализируем поток нулями
        total_flow = 0
        total_cost = 0.0

        # Основной цикл алгоритма
        while True:
            # Используем алгоритм Беллмана-Форда для нахождения кратчайшего пути
            dist = defaultdict(lambda: float('inf'))
            parent = {}
            dist[source] = 0

            # Релаксация рёбер
            for _ in range(len(self.nodes) - 1):
                for u in self.nodes:
                    for v in self.nodes:
                        if (self.graph[u][v] > self.flow_with_cost[u][v] and
                            dist[u] != float('inf') and
                            dist[u] + self.cost[u][v] < dist[v]):
                            dist[v] = dist[u] + self.cost[u][v]
                            parent[v] = u

            # Если путь до стока не найден, завершаем
            if dist[sink] == float('inf'):
                break

            # Находим минимальный поток на пути
            path_flow = float('inf')
            s = sink
            while s != source:
                u = parent[s]
                path_flow = min(path_flow, self.graph[u][s] - self.flow_with_cost[u][s])
                s = u

            # Обновляем поток и стоимость
            total_flow += path_flow
            v = sink
            while v != source:
                u = parent[v]
                self.flow_with_cost[u][v] += path_flow
                self.flow_with_cost[v][u] -= path_flow
                total_cost += path_flow * self.cost[u][v]
                v = u
```

```
            return total_flow, total_cost

    # Демонстрация работы
    def main():
        print("="*70)
        print("АНАЛИЗ ГАЗОТРАНСПОРТНОЙ СИСТЕМЫ РОССИИ")
        print("С УЧЕТОМ ОГРАНИЧЕНИЙ И СТОИМОСТИ ТРАНСПОРТИРОВКИ")
        print("="*70)

        # Создаем сеть с ограничениями на узлы
        network = GasTransportNetwork()

        # Устанавливаем ограничения на пропускную способность узлов
        network.set_node_capacity("КС_Муравленко", 200)
        network.set_node_capacity("КС_Новый_Уренгой", 250)
        network.set_node_capacity("Магистраль_Центральная", 300)

        print("\n1. ПОСТРОЕНИЕ СЕТИ С ОГРАНИЧЕНИЯМИ НА УЗЛЫ")
        print("-"*70)

        # Добавляем рёбра с учетом ограничений на узлы
        network.add_edge("ИСТОЧНИК", "Месторождение_Уренгой", 330)
        network.add_edge("ИСТОЧНИК", "Месторождение_Ямбург", 120)

        # Добавляем рёбра с ограничениями на узлы
        network.add_edge_with_node_constraints("Месторождение_Уренгой", "КС_Муравленко", 180)
        network.add_edge_with_node_constraints("Месторождение_Уренгой", "КС_Новый_Уренгой", 150)
        network.add_edge_with_node_constraints("Месторождение_Ямбург", "КС_Новый_Уренгой", 120)

        network.add_edge_with_node_constraints("КС_Муравленко", "Магистраль_Центральная", 160)
        network.add_edge_with_node_constraints("КС_Новый_Уренгой", "Магистраль_Центральная", 190)
        network.add_edge_with_node_constraints("КС_Новый_Уренгой", "Магистраль_Альтернативная", 100)

        network.add_edge("Магистраль_Центральная", "Потребители_Европа", 200)
        network.add_edge("Магистраль_Центральная", "Потребители_Азия", 150)
        network.add_edge("Магистраль_Альтернативная", "Потребители_Азия", 120)

        network.add_edge("Потребители_Европа", "СТОК", 200)
        network.add_edge("Потребители_Азия", "СТОК", 270)

        # Вычисляем максимальный поток
        max_flow = network.ford_fulkerson("ИСТОЧНИК", "СТОК")

        print(f"\nМаксимальный поток с учетом ограничений: {max_flow} млрд м³/год")

        # Создаем интерактивную визуализацию
        print("\n2. СОЗДАНИЕ ИНТЕРАКТИВНОЙ ДИАГРАММЫ")
        print("-"*70)

        fig = network.create_interactive_visualization("ИСТОЧНИК", "СТОК", max_flow)
        fig.show()

        # Демонстрация сети со стоимостью
        print("\n3. АНАЛИЗ С УЧЕТОМ СТОИМОСТИ ТРАНСПОРТИРОВКИ")
        print("-"*70)

        cost_network = CostFlowNetwork()

        # Добавляем рёбра со стоимостью транспортировки
        cost_network.add_edge_with_cost("ИСТОЧНИК", "Месторождение_Уренгой", 330, 50)
        cost_network.add_edge_with_cost("ИСТОЧНИК", "Месторождение_Ямбург", 120, 70)
        cost_network.add_edge_with_cost("Месторождение_Уренгой", "КС_Муравленко", 180, 30)
        cost_network.add_edge_with_cost("Месторождение_Уренгой", "КС_Новый_Уренгой", 150, 25)
        cost_network.add_edge_with_cost("Месторождение_Ямбург", "КС_Новый_Уренгой", 120, 40)
        cost_network.add_edge_with_cost("КС_Муравленко", "Магистраль_Центральная", 160, 20)
        cost_network.add_edge_with_cost("КС_Новый_Уренгой", "Магистраль_Центральная", 190, 15)
        cost_network.add_edge_with_cost("КС_Новый_Уренгой", "Магистраль_Альтернативная", 100, 35)
        cost_network.add_edge_with_cost("Магистраль_Центральная", "Потребители_Европа", 200, 60)
        cost_network.add_edge_with_cost("Магистраль_Центральная", "Потребители_Азия", 150, 80)
        cost_network.add_edge_with_cost("Магистраль_Альтернативная", "Потребители_Азия", 120, 75)
        cost_network.add_edge_with_cost("Потребители_Европа", "СТОК", 200, 10)
        cost_network.add_edge_with_cost("Потребители_Азия", "СТОК", 270, 12)

        # Вычисляем максимальный поток минимальной стоимости
        max_flow_cost, total_cost = cost_network.min_cost_max_flow("ИСТОЧНИК", "СТОК")

        print(f"Максимальный поток: {max_flow_cost} млрд м³/год")
        print(f"Общая стоимость транспортировки: {total_cost:,.0f} руб./год")
        print(f"Средняя стоимость: {total_cost/max_flow_cost:.1f} руб./млрд м³")

    if __name__ == "__main__":
        main()
```

```
======================================================================
АНАЛИЗ ГАЗОТРАНСПОРТНОЙ СИСТЕМЫ РОССИИ
С УЧЕТОМ ОГРАНИЧЕНИЙ И СТОИМОСТИ ТРАНСПОРТИРОВКИ
======================================================================

1. ПОСТРОЕНИЕ СЕТИ С ОГРАНИЧЕНИЯМИ НА УЗЛЫ
----------------------------------------------------------------------

Максимальный поток с учетом ограничений: 0 млрд м³/год

2. СОЗДАНИЕ ИНТЕРАКТИВНОЙ ДИАГРАММЫ
----------------------------------------------------------------------
```




```
3. АНАЛИЗ С УЧЕТОМ СТОИМОСТИ ТРАНСПОРТИРОВКИ
----------------------------------------------------------------------
Максимальный поток: 430 млрд м³/год
Общая стоимость транспортировки: 80,860 руб./год
Средняя стоимость: 188.0 руб./млрд м³
```

задача 4

Напишите программный код или сгенерируйте его с помощью искусственного интеллекта.