Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

**ЛАБОРАТОРНАЯ РАБОТА №2**
по дисциплине «**Моделирование**»

Выполнил:
студент гр. ИС-142
«__» мая 2025 г. _____ /Наумов А.А./

Проверил:
преподаватель
«__» мая 2025 г. _____ /Уженцева А.В./

Оценка « _____»

Новосибирск 2025

# ВВЕДЕНИЕ

В данной работе была рассмотрена задача нахождения критического пути во взвешенном ориентированном графе. Эта задача применяется в управлении проектами, сетевом планировании и анализе временных характеристик сложных систем. Основная цель работы заключалась в вычислении параметров для каждой дуги графа, таких как раннее и позднее время начала и окончания работ, а также резервов времени. Итогом выполнения стал расчет критического пути и его стоимости.

# ВЫПОЛНЕНИЕ РАБОТЫ

Задан ориентированный взвешенный граф, представленный списком рёбер в формате (начальная вершина – конечная вершина – вес ребра). Для него была проведена обработка по следующему алгоритму:

1. **Построение графа**: Граф задавался в виде матрицы смежности или списка ребер.

2. **Определение ранних сроков событий:**
   - Для стартовых рёбер (исходящих из начальной вершины) раннее начало t_РН равно 0.
   - Раннее окончание t_РО вычисляется как сумма t_РН и веса дуги.
   - Для других ребер t_РН определяется как максимальное значение t_РО предшествующих вершин.

3. **Определение поздних сроков событий:**
   - Для конечных рёбер позднее окончание t_ПО принимается равным стоимости критического пути.
   - Позднее начало t_ПН вычисляется как разность t_ПО и веса дуги.
   - Для других ребер t_ПО определяется как минимальное значение t_ПН последующих вершин.

4. **Расчет резервов времени:**
   - Полный резерв R вычисляется как разность позднего и раннего времени ( $R = t\_ПН - t\_РН$).
   - Локальный резерв r определяется как разность t_РН следующего самого дорогого ребра и t_РО текущего ребра.

5. **Поиск критического пути:**
   - Критический путь представляет собой самую длинную последовательность работ от начальной вершины до конечной.

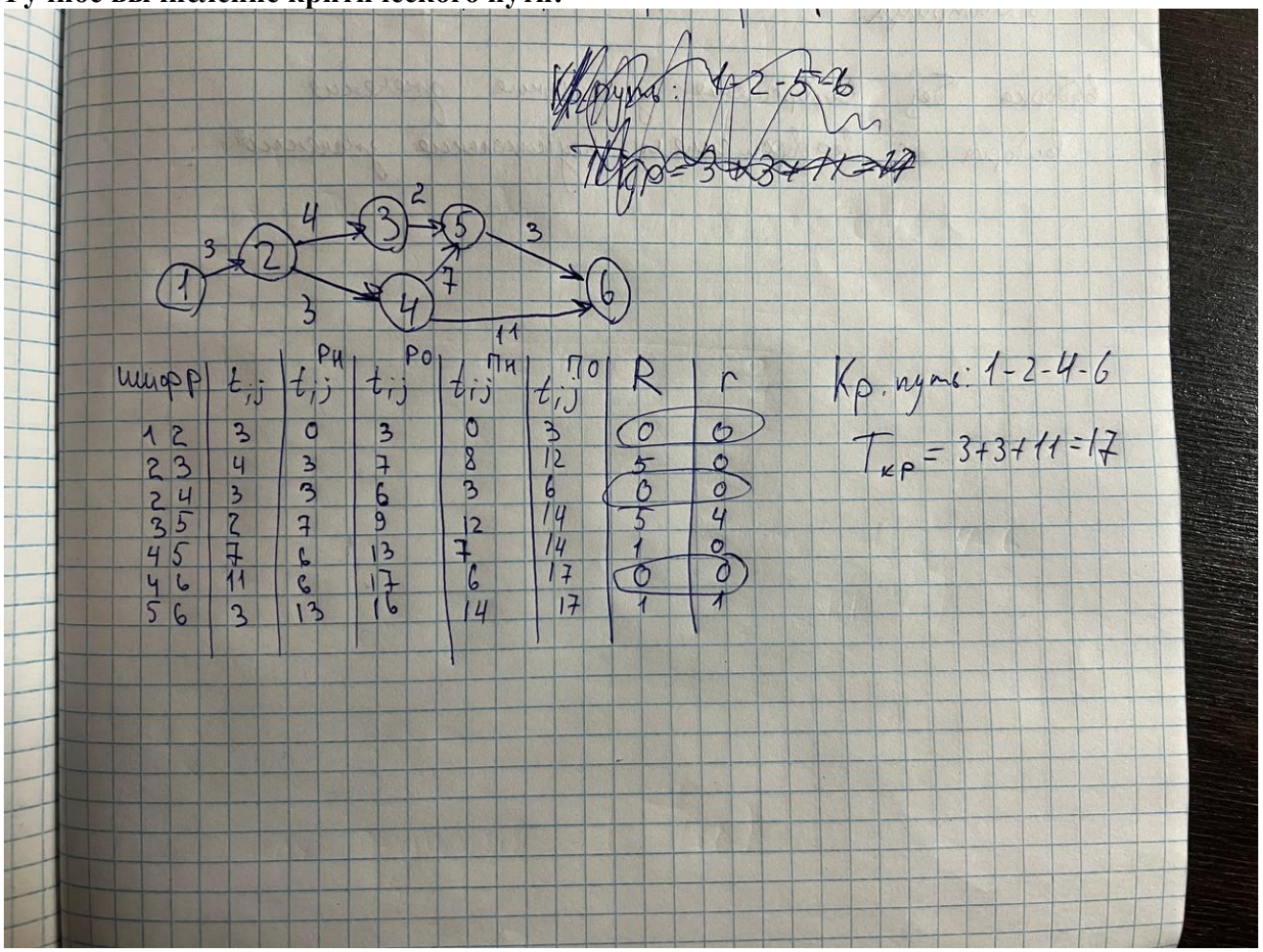
- Стоимость критического пути определяется как сумма весов его ребер.

6. **Визуализация графа:**
   - Граф строился с использованием библиотеки Matplotlib и NetworkX для наглядного представления структуры и критического пути.

**Дан граф :**

- 1-2: с весом 3
- 2-3: с весом 4
- 2-4: с весом 3
- 3-5: с весом 2
- 4-5: с весом 7
- 4-6: с весом 11
- 5-6: с весом 3

**Ручное вычисление критического пути:**



## Листинг программы:

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import scipy as sp

def calculate_schedule_parameters(G):
    """
    Вычисляем временные параметры:
      - early_start (ES) для каждой вершины
      - early_finish (EF): ES + (макс. вес исходящих ребер, если есть, иначе 0)
      - Поздние показатели вычисляем обратным проходом,
        latest_finish (LF) и latest_start (LS)
      - Для ребра (u,v):
            R = LS(v) - ES(u)
            r = (min(ES для всех потомков v) или EF(v), если потомков нет) - EF(u)
    """
    # Топологическая сортировка
    top_order = list(nx.topological_sort(G))

    # Вычисляем раннее начало (ES)
    early_start = {node: 0 for node in G.nodes()}
    for node in top_order:
        for succ in G.successors(node):
            weight = G[node][succ]['weight']
            early_start[succ] = max(early_start[succ], early_start[node] + weight)

    # Вычисляем раннее окончание (EF)
    early_finish = {}
    for node in G.nodes():
        outgoing = [G[node][succ]['weight'] for succ in G.successors(node)]
        early_finish[node] = early_start[node] + (max(outgoing) if outgoing else 0)

    # Инициализация позднего окончания (LF) – для последней вершины берём EF последней вершины
    last_node = top_order[-1]
    latest_finish = {node: early_finish[last_node] for node in G.nodes()}
    for node in reversed(top_order):
        for pred in G.predecessors(node):
            weight = G[pred][node]['weight']
            latest_finish[pred] = min(latest_finish[pred], latest_finish[node] - weight)

    # Вычисляем позднее начало (LS)
    latest_start = {}
    for node in G.nodes():
        incoming = [G[pred][node]['weight'] for pred in G.predecessors(node)]
        latest_start[node] = latest_finish[node] - (max(incoming) if incoming else 0)

    # Резервы для каждого ребра (u,v)
    reserves = {}
    local_reserves = {}
    for u, v in G.edges():
        reserves[(u, v)] = latest_start[v] - early_start[u]
        successors = list(G.successors(v))
        if successors:
            local_min = min(early_start[succ] for succ in successors)
        else:
            local_min = early_finish[v]
        local_reserves[(u, v)] = local_min - early_finish[u]

    return early_start, early_finish, latest_start, latest_finish, reserves, local_reserves

def calculate_critical_paths(G, source, target):
    """
    Находим критическую стоимость (максимальный суммарный вес от source до target)
    и перебираем все простые пути, сумма весов которых равна этой стоимости.
    """
    # Вычисляем раннее начало для определения критической стоимости
    top_order = list(nx.topological_sort(G))
    es = {node: float('-inf') for node in G.nodes()}
    es[source] = 0
    for node in top_order:
        for succ in G.successors(node):
```

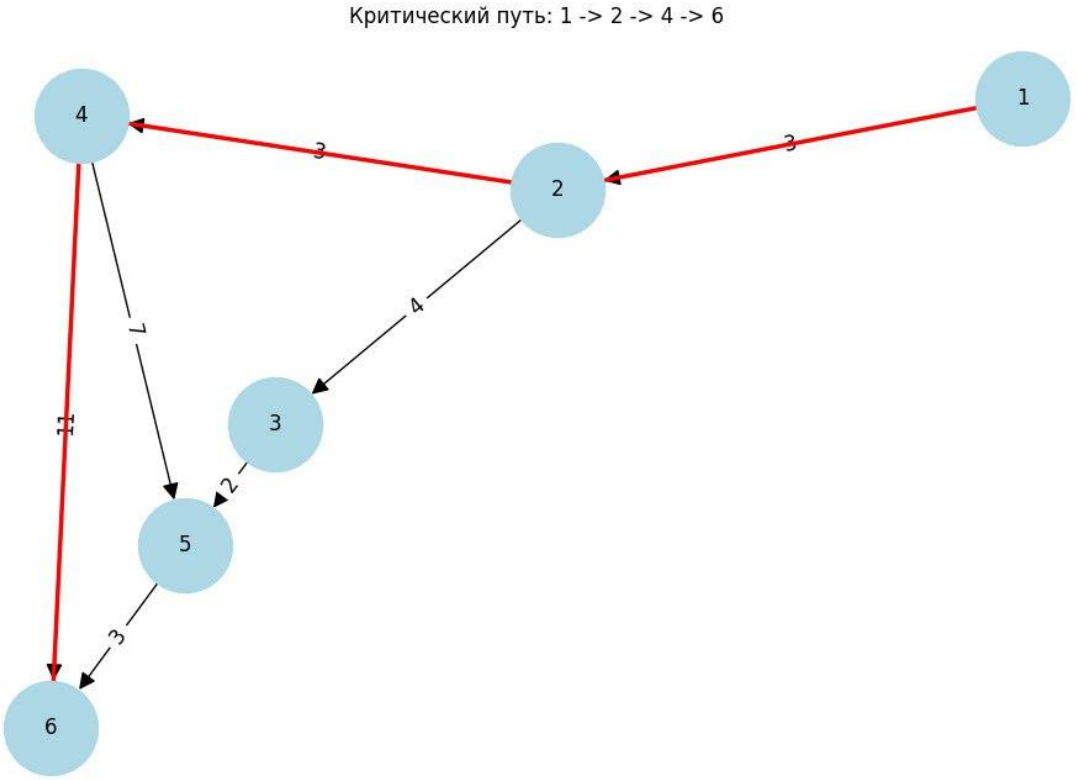
```
            weight = G[node][succ]['weight']
            es[succ] = max(es[succ], es[node] + weight)
    critical_cost = es[target]

    # Перебор всех простых путей от source до target
    all_paths = list(nx.all_simple_paths(G, source, target))
    critical_paths = []
    for path in all_paths:
        total = sum(G[path[i]][path[i+1]]['weight'] for i in range(len(path)-1))
        if total == critical_cost:
            critical_paths.append(path)
    return critical_cost, critical_paths


def plot_graphs_with_critical_paths(G, pos, critical_paths):
    """
    Строит граф для каждого критического пути, выделяя его ребра красным.
    """
    n = len(critical_paths)
    fig, axes = plt.subplots(1, n, figsize=(10 * n, 8))  # Увеличен размер
    if n == 1:
        axes = [axes]

    for ax, path in zip(axes, critical_paths):
        # Увеличен размер узлов и стрелок
        nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='black',
                node_size=3000, font_size=12, ax=ax, arrowsize=20)
        edge_labels = {(u, v): f"{G[u][v]['weight']}" for u, v in G.edges()}
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=12, ax=ax)
        path_edges = list(zip(path, path[1:]))
        nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color="red", width=2.5, ax=ax)
        ax.set_title("Критический путь: " + " -> ".join(map(str, path)))

    plt.show()

# ---------- Основной блок программы ----------

# Задаём данные графа
edges = [
        (1, 2, 3),
        (2, 3, 4),
        (2, 4, 3),
        (3, 5, 2),
        (4, 5, 7),
        (4, 6, 11),
        (5, 6, 3)
    ]
source = 1
target = 6

# Создаём граф
G = nx.DiGraph()
for u, v, w in edges:
    G.add_edge(u, v, weight=w)

# Вычисляем параметры расписания
early_start, early_finish, latest_start, latest_finish, reserves, local_reserves = calculate_schedule_parameters(G)

# Формируем таблицу со всеми данными
data = []
for u, v, w in edges:
    data.append([
        f"{u}-{v}",
        w,
        early_start[u],
        early_finish[u],
        latest_start[u],
        latest_finish[u],
        reserves[(u, v)],
        local_reserves[(u, v)]
    ])
df = pd.DataFrame(data, columns=["Edge", "t_ij", "t_ij_RN", "t_ij_RO", "t_ij_PN", "t_ij_PO", "R", "r"])

# Находим критические пути (те, сумма весов которых равна критической стоимости)
```

```
critical_cost, critical_paths = calculate_critical_paths(G, source, target)
print("\nКритическая стоимость (максимальный суммарный вес):", critical_cost)
print("Найденные критические пути:")
for path in critical_paths:
    print(" -> ".join(map(str, path)))

# Для построения графов используем общее расположение вершин
pos = nx.kamada_kawai_layout(G)

# Рисуем графы с критическими путями (на каждом subplot свой путь)
plot_graphs_with_critical_paths(G, pos, critical_paths)
```

```
alexeynaumov@Lenovo-Legion-5:~/4course/model$ python3 1.py

Критическая стоимость (максимальный суммарный вес): 17
Найденные критические пути:
1 -> 2 -> 4 -> 6
```

**Таблица с результатом:**

| Edge | t_ij | t_ij_RN | t_ij_RO | t_ij_PN | t_ij_PO | R | r |
|---|---|---|---|---|---|---|---|
| 1-2 | 3 | 0 | 3 | 0 | 3 | 0 | 0 |
| 2-3 | 4 | 3 | 7 | 8 | 12 | 5 | 0 |
| 2-4 | 3 | 3 | 6 | 3 | 6 | 0 | 0 |
| 3-5 | 2 | 7 | 9 | 12 | 14 | 5 | 4 |
| 4-5 | 7 | 6 | 13 | 7 | 14 | 1 | 0 |
| 4-6 | 11 | 6 | 17 | 6 | 17 | 0 | 0 |
| 5-6 | 3 | 13 | 16 | 14 | 17 | 1 | 1 |




## ЗАКЛЮЧЕНИЕ

В ходе работы были успешно рассчитаны параметры временных характеристик для каждого ребра графа. Был найден критический путь, представляющий собой наиболее продолжительную последовательность работ, от которого зависит минимальное время выполнения всей задачи. Построение графа позволило визуализировать структуру зависимостей и убедиться в корректности проведенных расчетов.