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. _____ /Григорьев Ю.В./

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

Заданы два ориентированных взвешенных графа, представленных списком рёбер в формате (начальная вершина – конечная вершина – вес ребра). Для каждого из них 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.

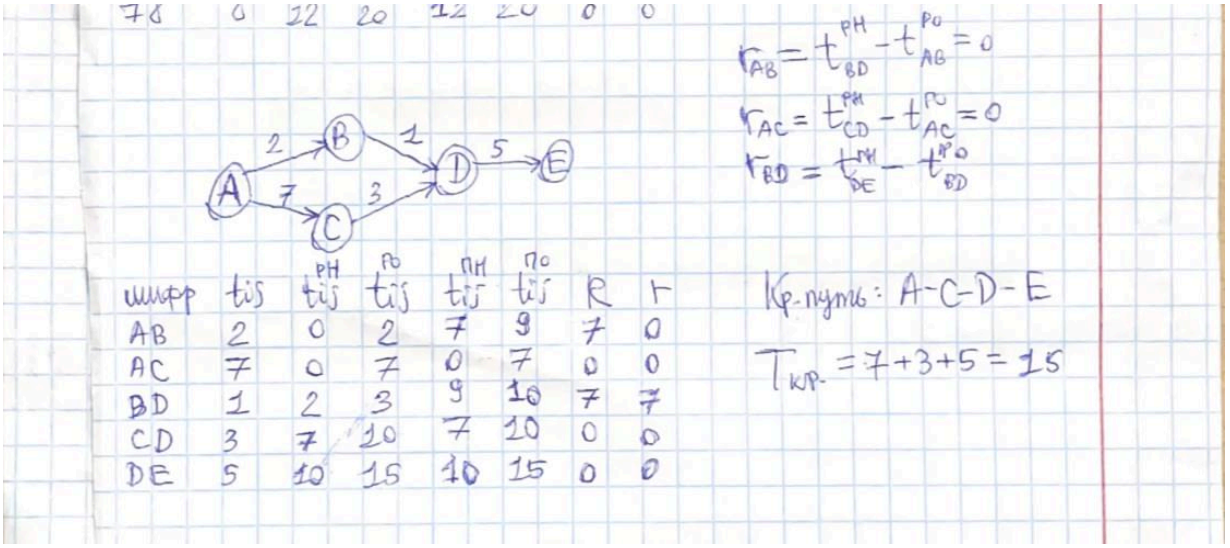
3. **Определение поздних сроков событий:**
   - Для конечных рёбер позднее окончание tПО принимается равным стоимости критического пути.
   - Позднее начало t_ПН вычисляется как разность t_ПО и веса дуги.
   - Для других ребер t_ПО определяется как минимальное значение t_ПН последующих вершин.

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

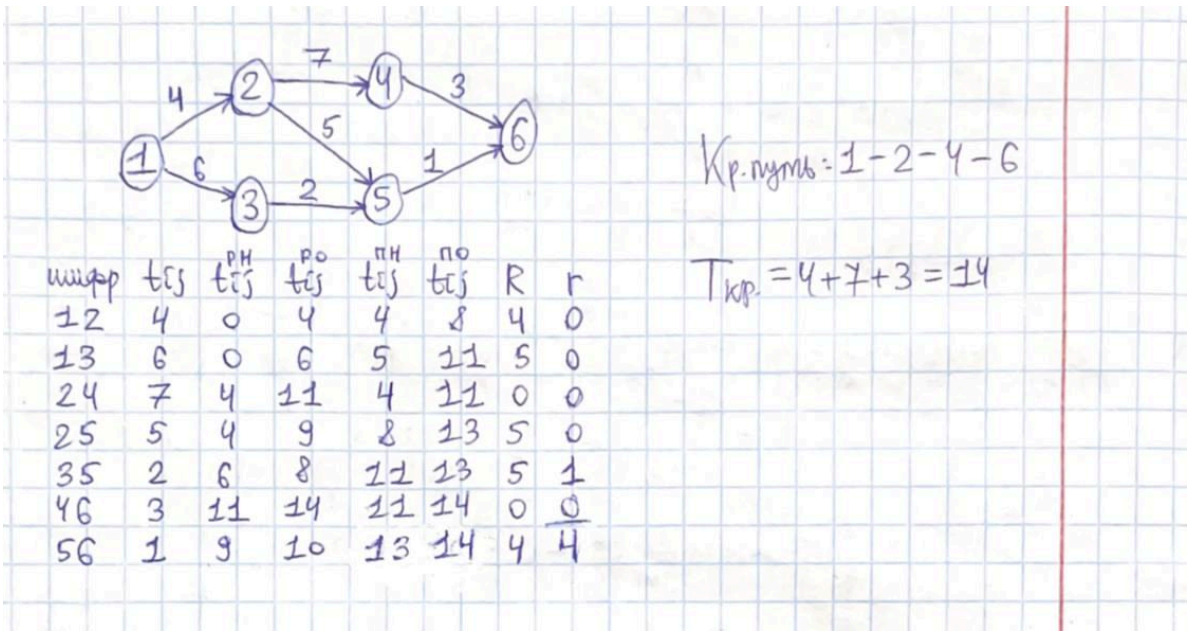
5. **Поиск критического пути:**
   - Критический путь состоит из ребер, у которых R=0.
   - Стоимость критического пути определяется как сумма весов его ребер.

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

Вычисления вручную



шифр	tij	tij РН	tij РО	tij ПН	tij ПО	R	r
AB	2	0	2	7	9	7	0
AC	7	0	7	0	7	0	0
BD	1	2	3	9	10	7	7
CD	3	7	10	7	10	0	0
DE	5	10	15	10	15	0	0

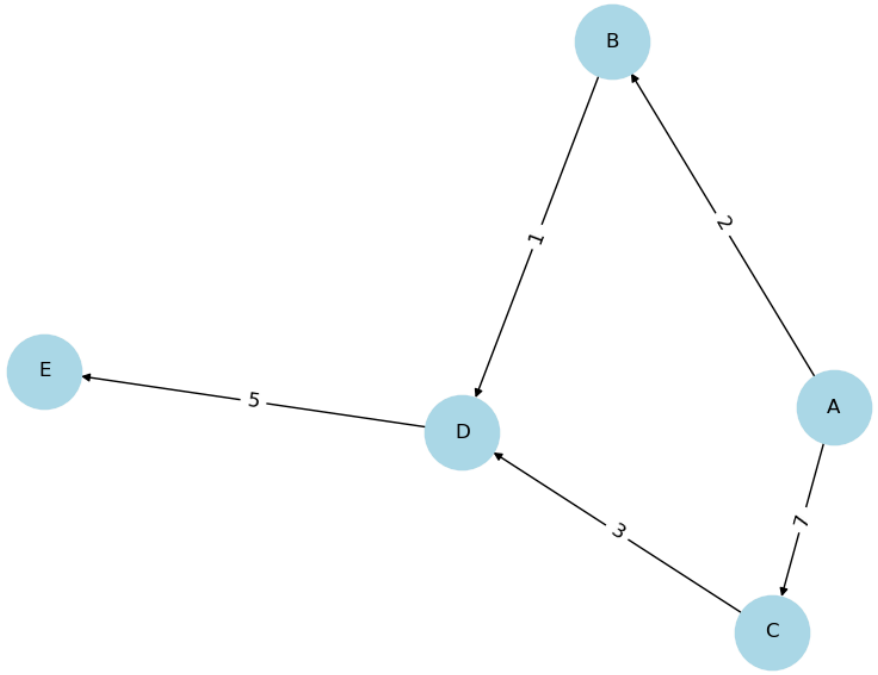


шифр	tij	tij РН	tij РО	tij ПН	tij ПО	R	r
12	4	0	4	4	8	4	0
13	6	0	6	5	11	5	0
24	7	4	11	4	11	0	0
25	5	4	9	8	13	5	0
35	2	6	8	11	13	5	1
46	3	11	14	11	14	0	0
56	1	9	10	13	14	4	4

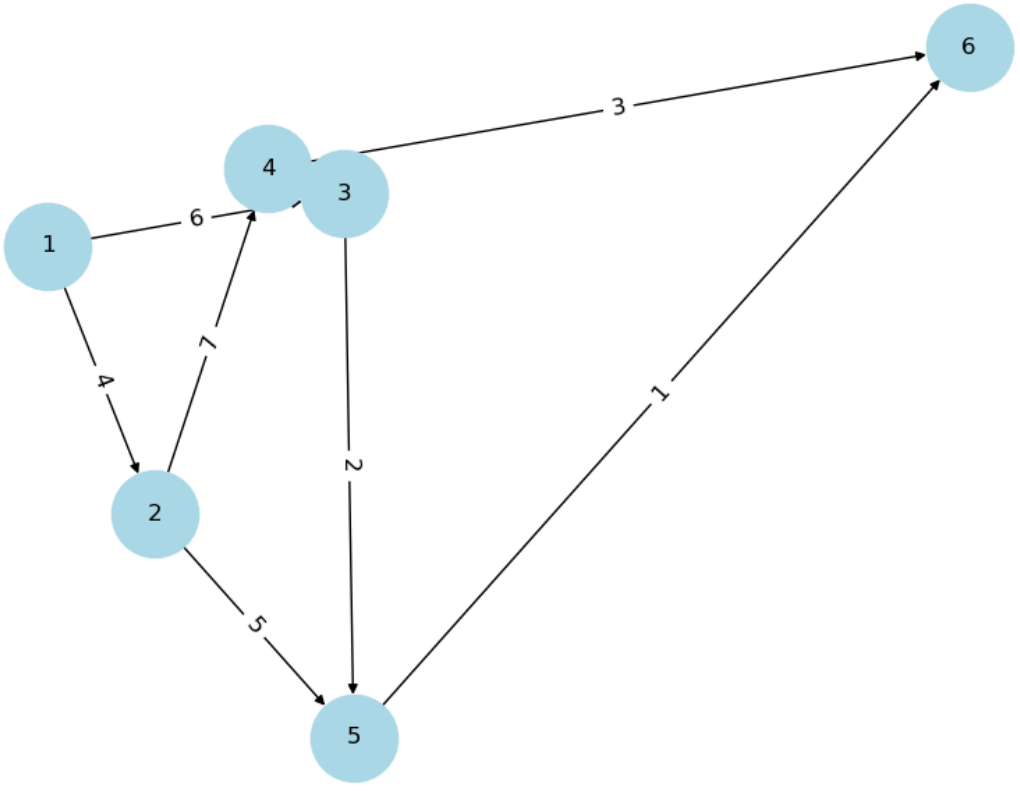
Программный код

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

def calculate_critical_path(edges):
    G = nx.DiGraph()
    for u, v, w in edges:
        G.add_edge(u, v, weight=w)

    # Топологическая сортировка
    top_order = list(nx.topological_sort(G))
```

```
    # Раннее начало и окончание
    early_start = {node: 0 for node in G.nodes()}
    early_finish = {}
    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)
    for node in G.nodes():
        early_finish[node] = early_start[node] + max((G[node][succ]['weight'] for succ in
G.successors(node)), default=0)

    # Позднее окончание и начало
    last_node = top_order[-1]
    latest_finish = {node: early_finish[last_node] for node in G.nodes()}
    latest_start = {}
    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)
    for node in G.nodes():
        latest_start[node] = latest_finish[node] - max((G[pred][node]['weight'] for pred in
G.predecessors(node)), default=0)

    # Резервы
    reserves = {}
    local_reserves = {}
    for u, v in G.edges():
        weight = G[u][v]['weight']
        reserves[(u, v)] = latest_start[v] - early_start[u]
        local_reserves[(u, v)] = min((early_start[succ] for succ in G.successors(v)),
default=early_finish[v]) - early_finish[u]

    # Критический путь
    critical_path = [edge for edge in G.edges() if reserves[edge] == 0]
    critical_cost = sum(G[u][v]['weight'] for u, v in critical_path)

    # Вывод таблицы
    data = []
    for u, v in G.edges():
        w = G[u][v]['weight']
        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"])
    print(df.to_string(index=False))
    print(f"\nCritical Path: {' -> '.join(str(u) for u, v in critical_path)}")
    print(f"Critical Path Cost: {critical_cost}")

    # Построение графа
    pos = nx.spring_layout(G)
    plt.figure(figsize=(8, 6))
    nx.draw(G, pos, with_labels=True, node_color='lightblue', edge_color='black', node_size=2000,
font_size=12)
    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)
    plt.show()

# Данные для двух графов
graph1 = [("A", "B", 2), ("A", "C", 7), ("B", "D", 1), ("C", "D", 3), ("D", "E", 5)]
graph2 = [(1, 2, 4), (1, 3, 6), (2, 4, 7), (2, 5, 5), (3, 5, 2), (4, 6, 3), (5, 6, 1)]

print("Graph 1:")
calculate_critical_path(graph1)
print("\nGraph 2:")
calculate_critical_path(graph2)
```



Граф 1.



Граф 2.

modelling > lab2 > out.txt

```
Graph 1:
Edge  t_ij  t_ij_RN  t_ij_RO  t_ij_PN  t_ij_PO  R  r
 A-B     2        0        2        7        9  7  0
 A-C     7        0        7        0        7  0  0
 B-D     1        2        3        9       10  7  7
 C-D     3        7       10        7       10  0  0
 D-E     5       10       15       10       15  0  0

Critical Path: A -> C -> D -> E
Critical Path Cost: 15

Graph 2:
Edge  t_ij  t_ij_RN  t_ij_RO  t_ij_PN  t_ij_PO  R  r
 1-2     4        0        4        4        8  4  0
 1-3     6        0        6        5       11  5  0
 2-4     7        4       11        4       11  0  0
 2-5     5        4        9        8       13  5  0
 3-5     2        6        8       11       13  5  1
 4-6     3       11       14       11       14  0  0
 5-6     1        9       10       13       14  4  4

Critical Path: 1 -> 2 -> 4 -> 6
Critical Path Cost: 14
```

Результаты в файле.

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 проведённых расчетов. Данный метод является эффективным инструментом для анализа временных затрат и оптимизации последовательности выполнения работ в проектировании и управлении процессами.