Задача 1. Составить сетевой график и определить продолжительность времени выполнения проекта. Данные для составления сетевого графика представлены в таблице 11.

Таблица 11 – Данные для построения сетевого графика

Процесс	Предшествующий процесс
A	-
B	A
C	A
D	A
E	C, D
F	E
G	D
H	G
I	H
J	F, I
K	F, I
L	J, K
M	L
N	L
O	D
P	B, M, N, O
Q	P
R	Q
S	Q
T	S
U	R, T

Продолжительность выполнения работ выбрать в соответствии с вариантами, представленными в таблице 12.

Таблица 12 – Варианты продолжительности выполнения работ

	2
A	11
B	12
C	18
D	20
E	17
F	17
G	12
H	13
I	13
J	11
K	20
L	11
M	11
N	20
O	14
P	20
Q	10
R	20
S	16
T	20
U	19

Решение

Начнем с составления графика проекта, определяя для каждого процесса предшествующие задачи и продолжительность выполнения. Затем найдем критический путь, чтобы определить общую продолжительность проекта.

Напишу Python-программу, которая строит сеть (используя библиотеку networkX), а также её вывод (используя библиотеку matplotlib)

Определяем узлы и длительности, создаем граф и добавляем к нему вершины и ребра. Затем найдем крит. путь и его длину

```
tasks = {
    "A": {"predecessors": [], "duration": 11},
    "B": {"predecessors": ["A"], "duration": 12},
    "C": {"predecessors": ["A"], "duration": 18},
    "D": {"predecessors": ["A"], "duration": 20},
    "E": {"predecessors": ["C", "D"], "duration": 17},
    "F": {"predecessors": ["E"], "duration": 17},
    "G": {"predecessors": ["D"], "duration": 12},
    "H": {"predecessors": ["G"], "duration": 13},
    "I": {"predecessors": ["H"], "duration": 13},
    "J": {"predecessors": ["F", "I"], "duration": 11},
    "K": {"predecessors": ["F", "I"], "duration": 20},
    "L": {"predecessors": ["J", "K"], "duration": 11},
    "M": {"predecessors": ["L"], "duration": 11},
    "N": {"predecessors": ["L"], "duration": 20},
    "O": {"predecessors": ["D"], "duration": 14},
    "P": {"predecessors": ["B", "M", "N", "O"], "duration": 20},
    "Q": {"predecessors": ["P"], "duration": 10},
    "R": {"predecessors": ["Q"], "duration": 20},
    "S": {"predecessors": ["Q"], "duration": 16},
    "T": {"predecessors": ["S"], "duration": 20},
    "U": {"predecessors": ["R", "T"], "duration": 19},
}

# Создаем направленный граф
G = nx.DiGraph()

# Добавляем вершины и ребра с весами
for task, details in tasks.items():
    for predecessor in details["predecessors"]:
        G.add_edge(predecessor, task, weight=details["duration"])

# Находим критический путь и его длину (общую продолжительность проекта)
critical_path = nx.dag_longest_path(G, weight="weight")
project_duration = nx.dag_longest_path_length(G, weight="weight")
```

Результат:

```
['A', 'D', 'G', 'H', 'I', 'K', 'L', 'N', 'P', 'Q', 'S', 'T', 'U']
194
['A - B = 12', 'A - C = 18', 'A - D = 20', 'B - P = 20', 'C - E = 17
11', 'I - K = 20', 'J - L = 11', 'K - L = 11', 'L - M = 11', 'L - N =
U = 19']
```

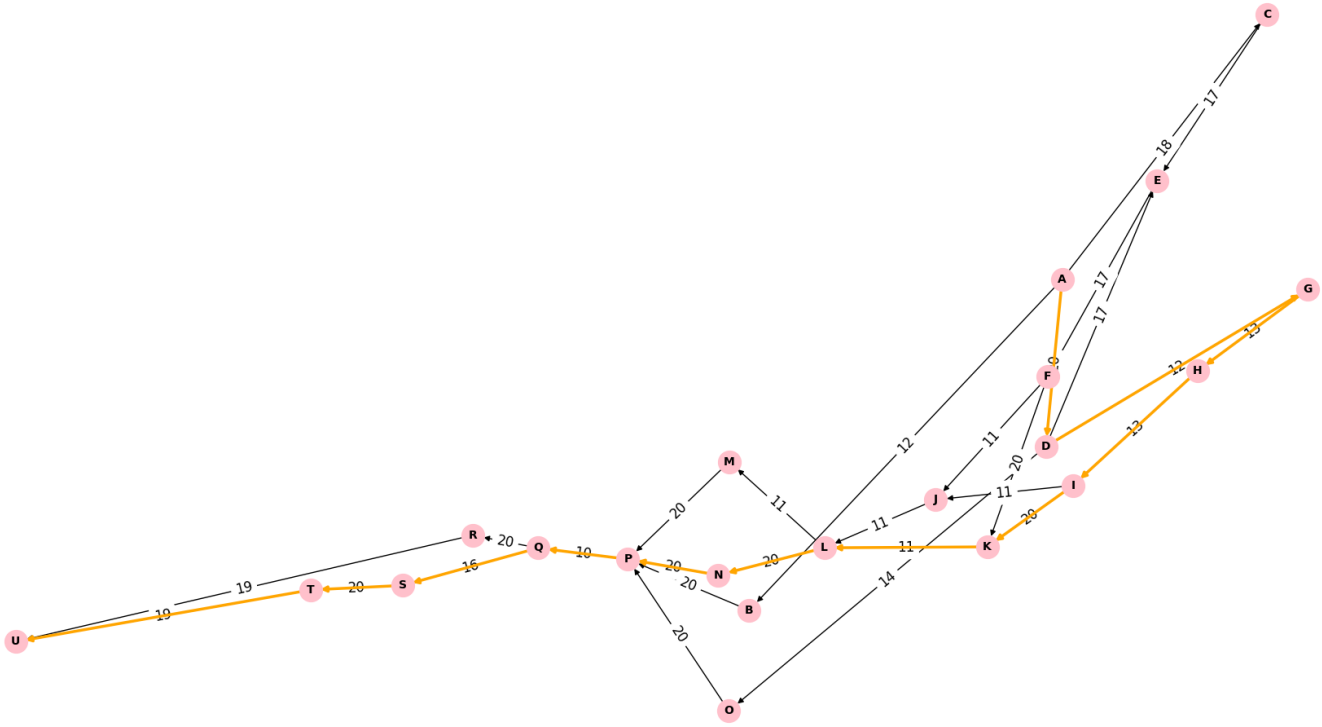
Вершины крит. пути: ['A', 'D', 'G', 'H', 'I', 'K', 'L', 'N', 'P', 'Q', 'S', 'T', 'U']
Длина крит. пути: 194

Весы ребер: ['A - B = 12', 'A - C = 18', 'A - D = 20', 'B - P = 20', 'C - E = 17', 'D - E = 17', 'D - G = 12', 'D - O = 14', 'E - F = 17', 'F - J = 11', 'F - K = 20', 'G - H = 13', 'H - I = 13', 'I - J = 11', 'I - K = 20', 'J - L = 11', 'K - L = 11', 'L - M = 11', 'L - N = 20', 'M - P = 20', 'N - P = 20', 'O - P = 20', 'P - Q = 10', 'Q - R = 20', 'Q - S = 16', 'R - U = 19', 'S - T = 20', 'T - U = 19']

Напишу код, который иллюстрирует граф и его крит путь графически.

```python
pos = nx.spring_layout(G, seed=42)

# Рисуем граф
plt.figure(figsize=(14, 10))
nx.draw(
    G,
    pos,
    with_labels=True,
    node_size=400,
    node_color="pink",
    font_size=10,
    font_weight="bold",
    arrows=True,
)

# Подписи к весам ребер (длительности задач)
edge_labels = {(u, v): data["weight"] for u, v, data in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=9)

# Подсветка критического пути
path_edges = list(zip(critical_path, critical_path[1:]))
nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color="orange", width=2.5)

plt.show()
```

Иллюстрация крит. пути:



Задача 2

Определить критический путь, полные независимые и свободные резервы некритических работ для сетевой модели, представленной на рисунке 6.

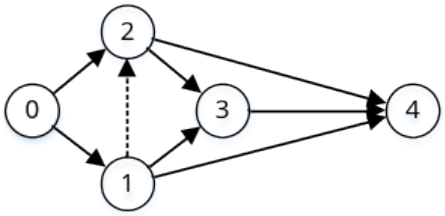


Рисунок 6 – Графическое представление сетевой модели

В качестве варианта использовать продолжительности работ, представленную в таблице 14.

Таблица 14 – Варианты продолжительности выполнения работ

Работа	4
0 – 1	3
0 – 2	4
1 – 3	5
2 – 3	3
2 – 4	7
3 – 4	3
1 – 4	6

Решение: сначала построим граф (сеть) и рассчитаем критический путь, а затем определим резервы для некритических работ.

Создадим соответствия вершин и весов ребер, объединим это в граф, найдем крит. путь и его длину.

```
# Создадим граф с учетом данных из таблицы
G = nx.DiGraph()

# Добавляем вершины и их длительности
edges_durations = {
    (0, 1): 4,
    (0, 2): 3,
    (1, 3): 4,
    (2, 3): 5,
    (2, 4): 3,
    (3, 4): 7,
    (1, 4): 3,
}

# Добавляем ребра в граф с указанием длительности как веса
for (start, end), duration in edges_durations.items():
    G.add_edge(start, end, weight=duration)

# Находим критический путь и его длину (общая продолжительность проекта)
critical_path = nx.dag_longest_path(G, weight="weight")
project_duration = nx.dag_longest_path_length(G, weight="weight")

# Рассчитываем ранние и поздние сроки для всех узлов, чтобы найти резервы
early_start = nx.dag_longest_path_length(G, weight="weight")
late_start = nx.dag_longest_path_length(G, weight="weight")

# Определим полные, независимые и свободные резервы для некритических работ
slack_info = {}
for u, v, data in G.edges(data=True):
    duration = data["weight"]
    early_start_u = nx.dag_longest_path_length(
        G.subgraph(nx.ancestors(G, u) | {u}), weight="weight"       Argument of type
    )
    early_start_v = nx.dag_longest_path_length(
        G.subgraph(nx.ancestors(G, v) | {v}), weight="weight"       Argument of type
    )
    slack_info[(u, v)] = {
        "duration": duration,
        "total_float": early_start_v - early_start_u - duration,
        "free_float": min(early_start_v - early_start_u - duration, duration),
    }

```

Результат:

```
Крит путь =  [0, 1, 3, 4]  =>  15
Полные независимые и свободные (free) резервы:  {(0,
ree_float': 0}, (0, 2): {'duration': 3, 'total_float
n': 4, 'total_float': 0, 'free_float': 0}, (1, 4): {
oat': 3}, (2, 3): {'duration': 5, 'total_float': 0,
 'total_float': 9, 'free_float': 3}, (3, 4): {'durati
0}}
```

Крит путь = [0, 1, 3, 4] => 15
Полные независимые и свободные (free) резервы: {(0, 1): {'duration': 4, 'total_float': 0, 'free_float': 0}, (0, 2): {'duration': 3, 'total_float': 0, 'free_float': 0}, (1, 3): {'duration': 4, 'total_float': 0, 'free_float': 0}, (1, 4): {'duration': 3, 'total_float': 8, 'free_float': 3}, (2,
 3): {'duration': 5, 'total_float': 0, 'free_float': 0}, (2, 4): {'duration': 3, 'total_float': 9, 'free_float': 3}, (3, 4): {'duration': 7, 'total_float': 0, 'free_float': 0}}

Иллюстрация крит. пути: