Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторня работа №9

Выполнил:
Стафеев И.А.
Проверил
Мусаев А.А.

Санкт-Петербург,
2024

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ

Цель работы: закрепить полученные за два семестра знания в области алгоритм и структур данных, решив несколько задач. Для достижения цели были поставлены следующие задачи:

- Написать гибридный алгоритм сортировки и сравнить его эффективность с другими;
- Написать жадный алгоритм для построения минимального остовного дерева в графе;
- Написать алгоритм поиска кратчайшего пути в графе.

1 Гибридный алгоритм сортировки

Необходимо написать гибридный алгоритм сортировки, использующий идеи сортировки вставками и сортировки слиянием. Код

Код алгоритма сортировки слиянием приведен на рисунке 1

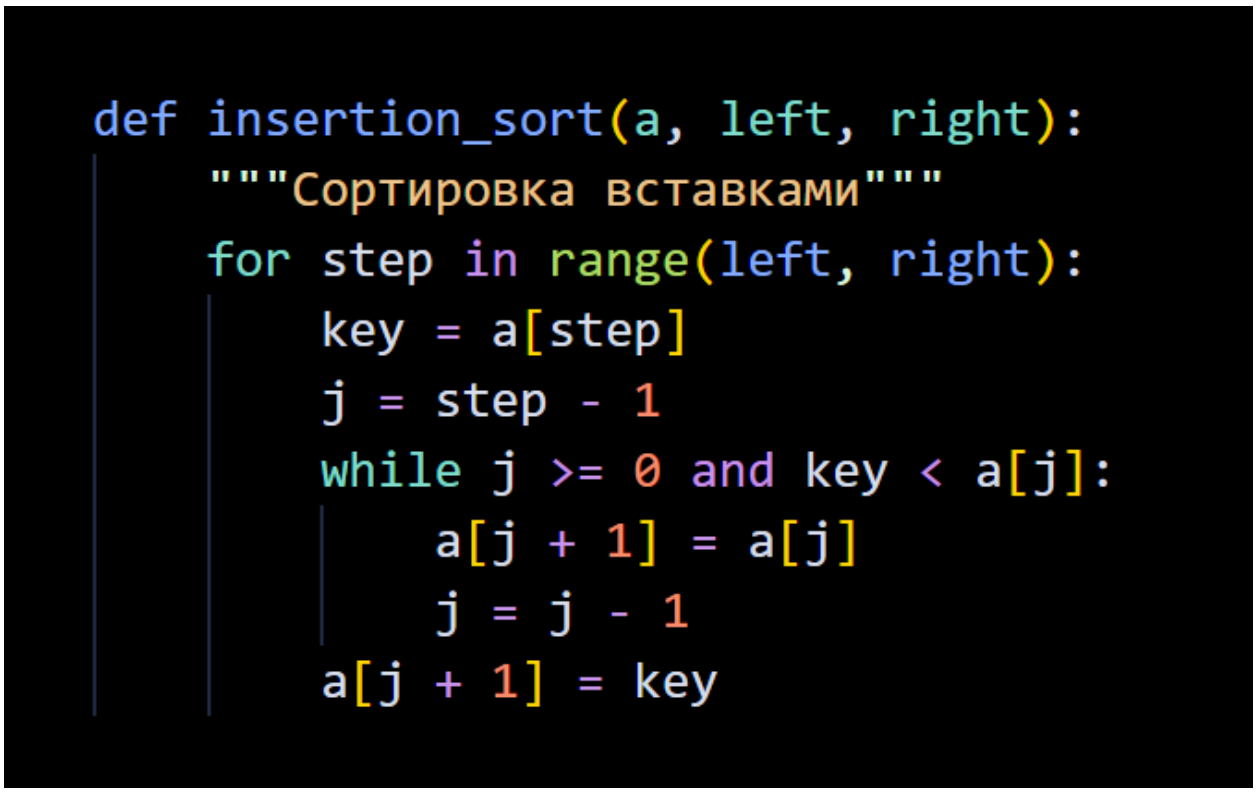

Рисунок 1 — Алгоритм сортировки вставками

Код функции слияния из алгоритма сортировки слиянием приведен на рисунке 2

```python
def merge(a: list[float], start: int, end: int, mid: int):
    """Слияние двух частей массива на очередном вызове рекурсии"""
    p1, p2 = 0, 0
    ix = start
    left_half, right_half = a[start:mid+1], a[mid+1:end+1]

    while p1 < len(left_half) and p2 < len(right_half):
        if left_half[p1] > right_half[p2]:
            a[ix] = right_half[p2]
            p2 += 1
        else:
            a[ix] = left_half[p1]
            p1 += 1
        ix += 1

    while p1 < len(left_half):
        a[ix] = left_half[p1]
        ix += 1
        p1 += 1

    while p2 < len(right_half):
        a[ix] = right_half[p2]
        ix += 1
        p2 += 1
```

Рисунок 2 — Функция слияния

Код гибридной сортировки приведен на рисунке 3. Идея заключается в том, что входные данные делятся на блоки размера *threshold*, которые сортируются вставками, а потом сливаются с помощью функции *merge*.

```python
def hybrid_sort(in_data, threshold=40):
    n = len(in_data)
    if n < threshold:
        insertion_sort(in_data, 0, n)
    for start in range(0, n, threshold):
        end = min(start + threshold - 1, n - 1)
        insertion_sort(in_data, start, end + 1)

    curr_size = threshold
    while curr_size < n:
        for start in range(0, n, curr_size * 2):
            mid = min(n - 1, start + curr_size - 1)
            end = min(n - 1, mid + curr_size)
            merge(in_data, start, end, mid)
        curr_size *= 2
    return in_data
```

Рисунок 3 — Алгоритм гибридной сортировки

Ниже приведены тесты гибридной сортировки, сортировки слиянием, вставками, пирамидальной и быстрой сортировки

```
Сортировка вставками выполнилась за 4e-05 с
PS D:\ProgrammingProjects\itmo_algos_labs> & "D:/Program Files/
abs/Lab9/task3.py
Размер массива - 40
Гибридная сортировка выполнилась за 0.00011 с
Быстрая сортировка выполнилась за 0.00023 с
Пирамидальная сортировка выполнилась за 0.00068 с
Сортировка слиянием выполнилась за 0.00057 с
Сортировка вставками выполнилась за 0.00026 с
Самой быстрой оказалась сортировка - гибридная
PS D:\ProgrammingProjects\itmo_algos_labs>
```

Рисунок 4 — Тесты сортировок при размере массива 40 элементов

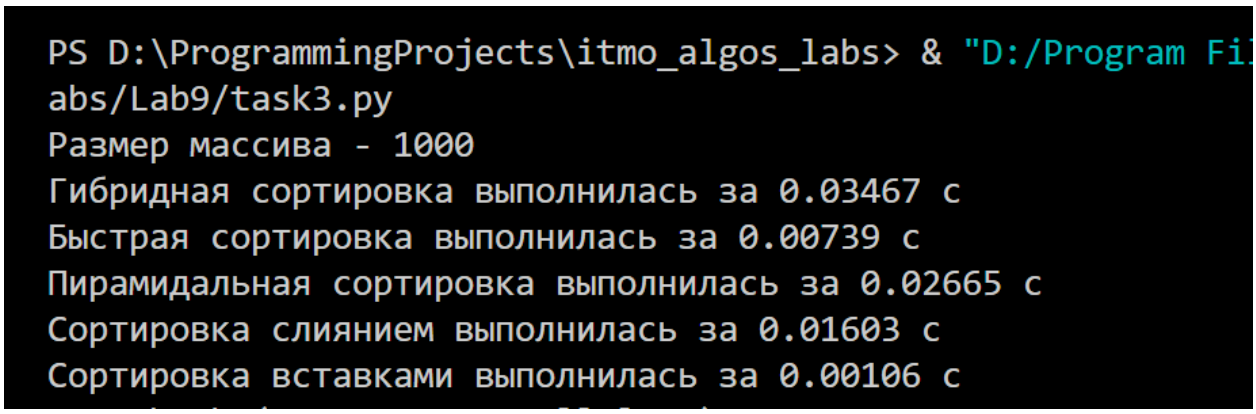

Рисунок 5 — Тесты сортировок при размере массива 1000 элементов

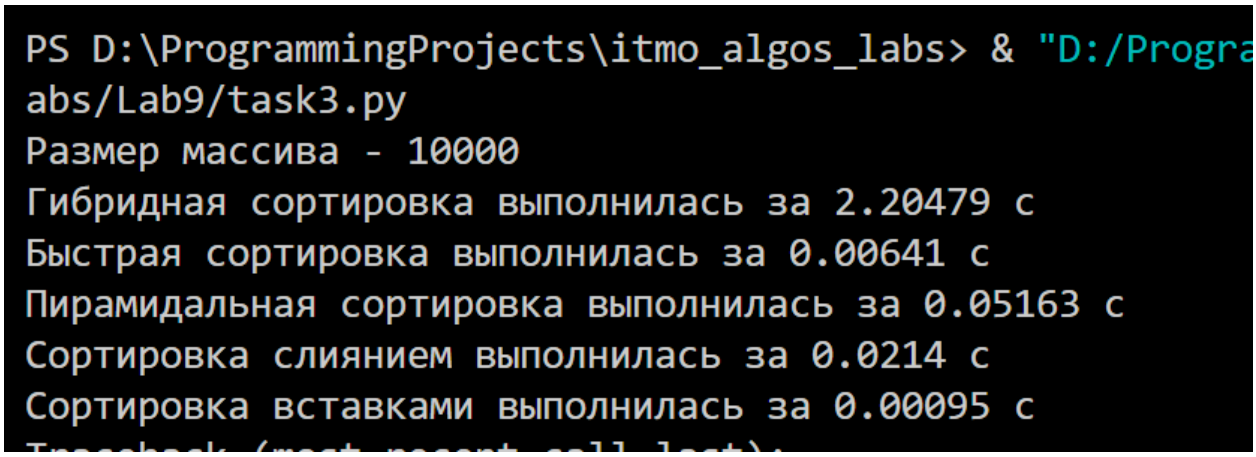

Рисунок 6 — Тесты сортировок при размере массива 10000 элементов

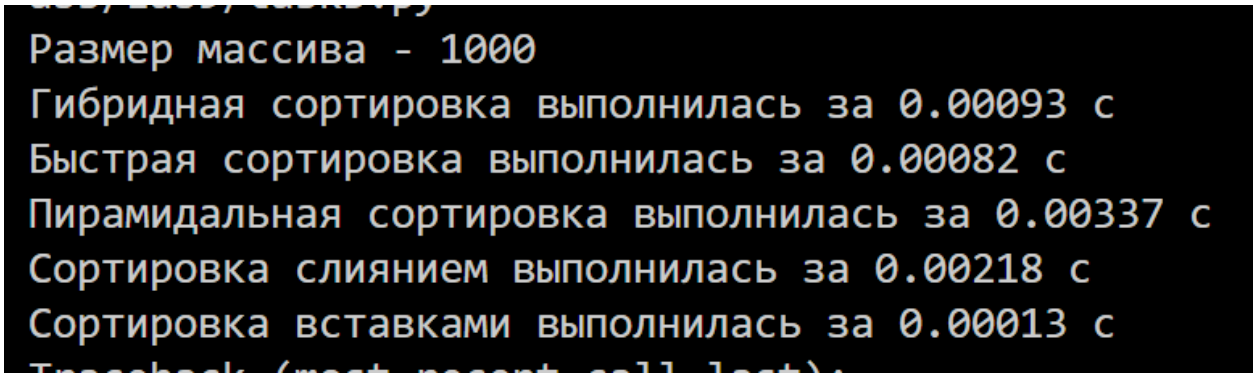

Рисунок 7 — Тесты сортировок при размере массива 1000 элементов, который уже отсортирован

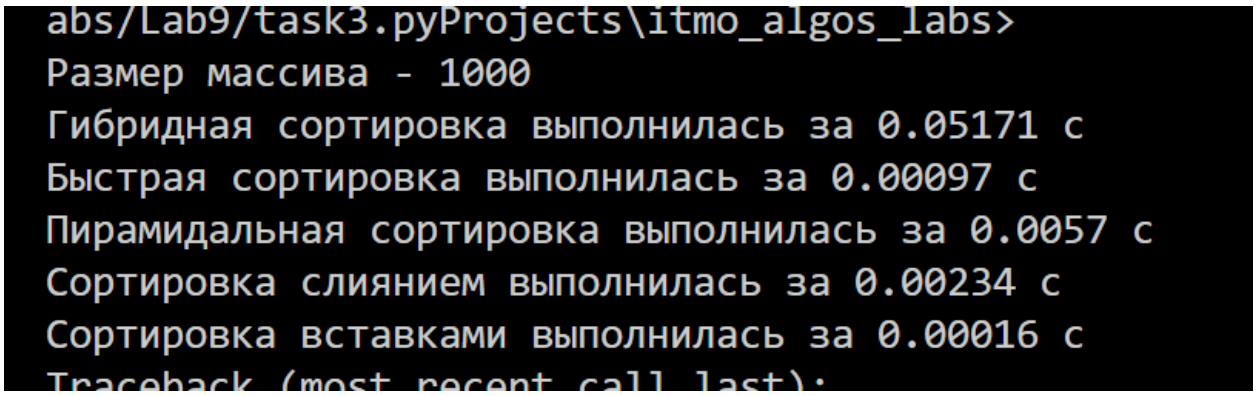

Рисунок 8 — Тесты сортировок при размере массива 1000 элементов, который отсортирован в обратном порядке

2 Поиск миимального остовного дерева

В работе был реализован алгоритм Крускала, предотвращающий образование циклов при построении остова. Код алгоритма приведен на рисунке 9

```python
def find_spanning_tree(g: list[tuple]) -> list[tuple]:
    """Реализация алгоритма Крускала"""
    nodes = set()
    edges = []
    ng = dict()
    for (u, v, w) in sorted(g, key=lambda x: x[2]):
        if u in nodes and v in nodes: # проверка на цикл
            continue
        if u not in nodes and v not in nodes:
            ng[u] = ng[v] = [u, v]
        else:
            if not ng.get(u):
                ng[v].append(u)
                ng[u] = ng[v]
            else:
                ng[u].append(v)
                ng[v] = ng[u]

        edges.append((u, v, w))
        nodes.add(u)
        nodes.add(v)

    for (u, v, w) in sorted(g, key=lambda x: x[2]): # объединение а
        if v not in ng[u]:
            edges.append((u, v, w))
            gr1 = ng[u]
            ng[u] += ng[v]
            ng[v] += gr1

    return edges
```

Рисунок 9 — Алгоритм Крускала

Пример работы алгоритма можно увидеть на рисунках 10 и 11.

```
PS D:\ProgrammingProjects\itmo_algos_labs> & "D:/Program Files/Python311/python.exe" d:/ProgrammingProjects/itmo_
abs/Lab9/task6.py
Минимальное остовное дерево имеет вес 39 и состоит из ребер [(1, 4), (3, 5), (4, 6), (1, 2), (5, 7), (2, 5)]
```

Рисунок 10 — Вывод алгоритм Крускала

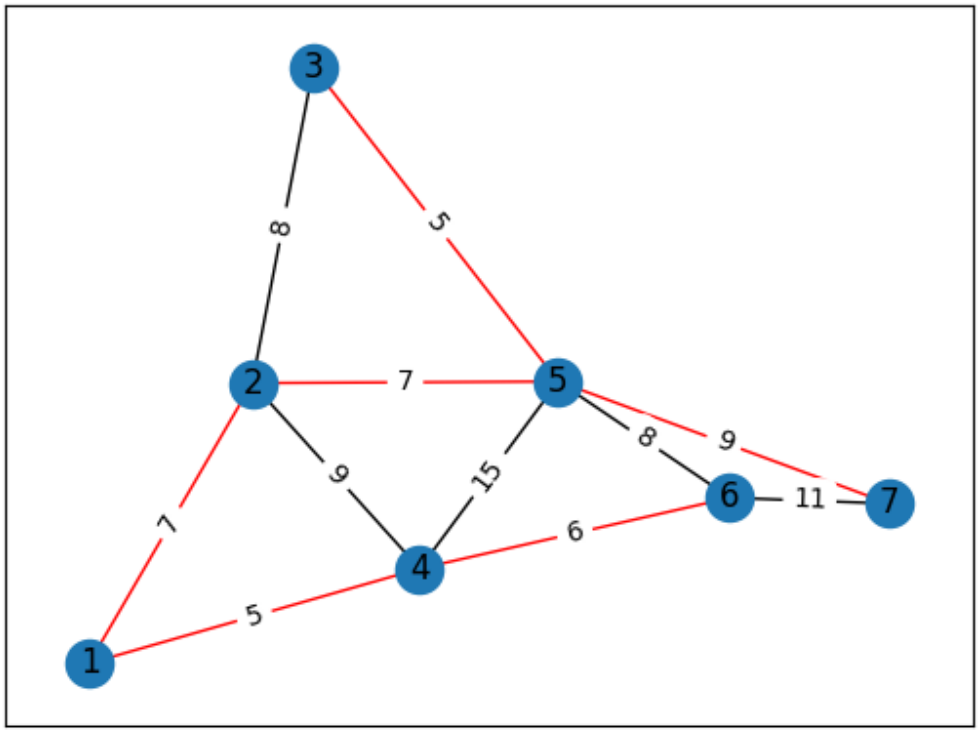

Рисунок 11 — Граф с выеделенным остовным деревом

3 Нахождение кратчайшего пути

Был реализован алгоритм Форда-Беллмана, его код приведен на рисунке 12.

```python
def find_shortest_path(start_vertex: int, end_vertex: int, num_of_vertexes: int,
                       num_of_edges: int, g: dict[int, list[dict]]) -> list[int]:
    """Реализация алгоритма Фора-Беллмана"""
    distances = [float("inf")] * (num_of_vertexes + 1)
    distances[start_vertex] = 0
    parents = dict()
    for _ in range(num_of_edges):
        for u in g:
            if distances[u] == float("inf"):
                continue
            for v, w in g[u].items():
                if distances[v] > distances[u] + w:
                    distances[v] = distances[u] + w
                    parents[v] = u
    # проверка на путь отрицательной длины
    for u in g:
        for v, w in g[u].items():
            if distances[u] != float("inf") and distances[v] > distances[u] + w:
                raise ValueError("Cycle with negative length found")
    path = []
    cur_vertex = end_vertex
    while cur_vertex != start_vertex:
        path.append(cur_vertex)
        cur_vertex = parents[cur_vertex]
    path.append(start_vertex)
    return distances[end_vertex], path[::-1]
```

Рисунок 12 — Алгоритм Форда-Беллмана

Результат работы для неорентированного графа приведен на рисунках 13 и 14.

```
PS D:\ProgrammingProjects\itmo_algos_labs> & "D:/Program
abs/Lab9/task9.py
Длина кратчайшего пути равна 20. Путь: 1->3->6->5
```

Рисунок 13 — Результат работы алгоритм для неорграфа

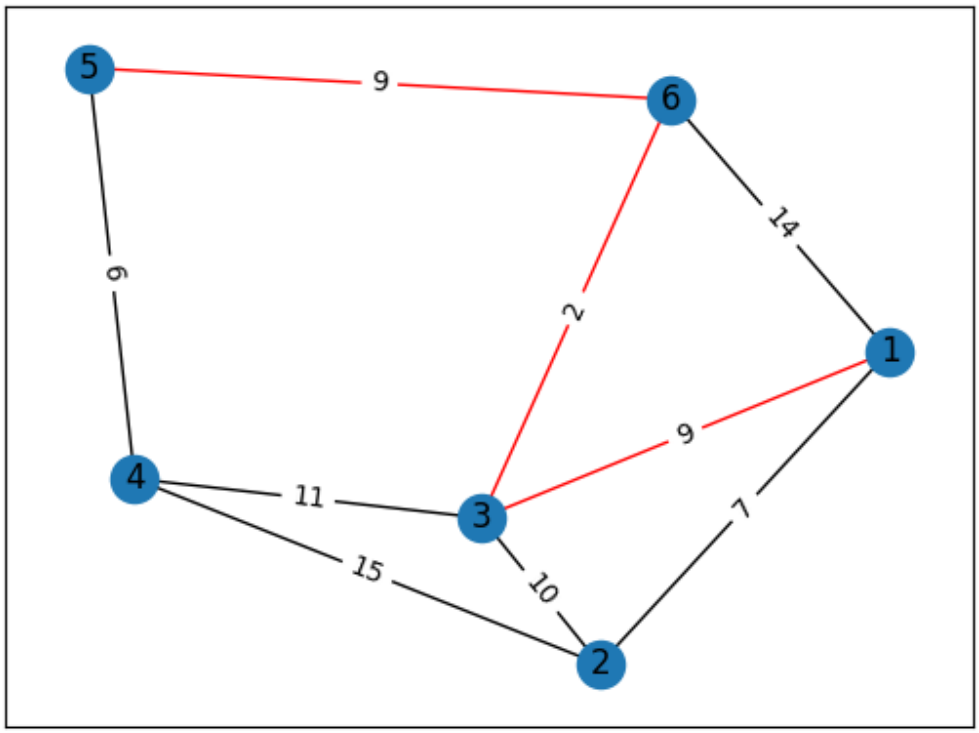

Рисунок 14 — Неорграф с выделенным путем

Результат работы для орентированного графа приведен на рисунках 13 и 14.

```
PS D:\ProgrammingProjects\itmo_algos_labs> & "D:/Program Files/Python311/
abs/Lab9/task9.py
Длина кратчайшего пути равна 15. Путь: 1->3->2->4->5->6
```

Рисунок 15 — Результат работы алгоритм для орграфа

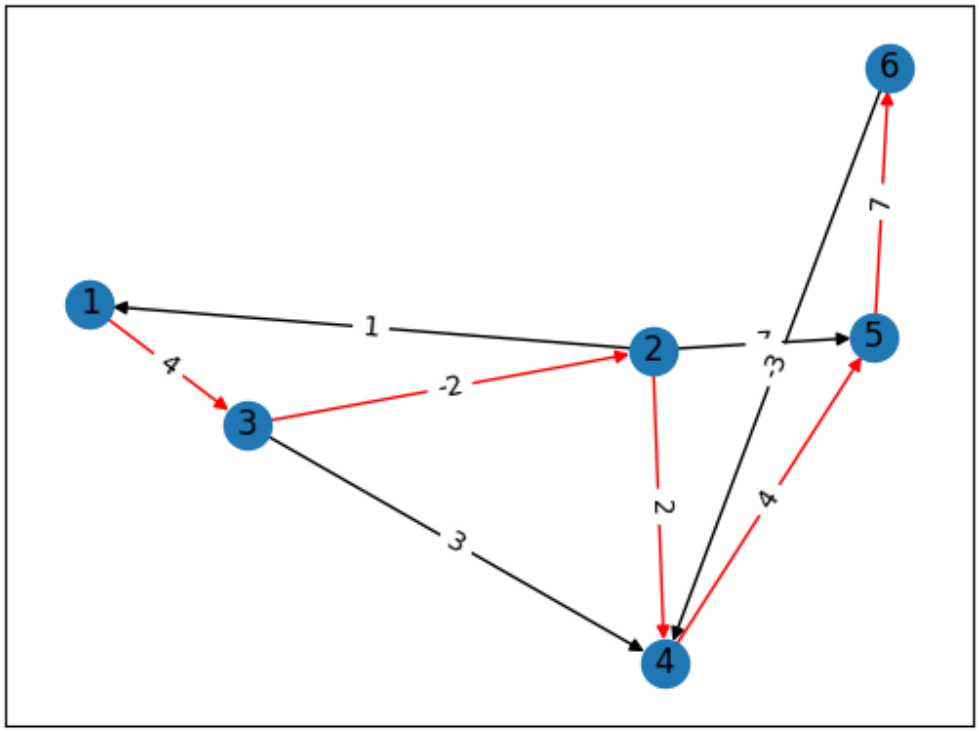

Рисунок 16 — Орграф с выделенным путем

Результат работы для графа с циклом отрицательного веса приведен на рисунках 17 и 18.

```
PS D:\ProgrammingProjects\itmo_algos_labs> & "D:/Prog
abs/Lab9/task9.py
В графе обнаружен цикл отрицательной длины
```

Рисунок 17 — Результат работы алгоритм для графа с отрицательным циклом

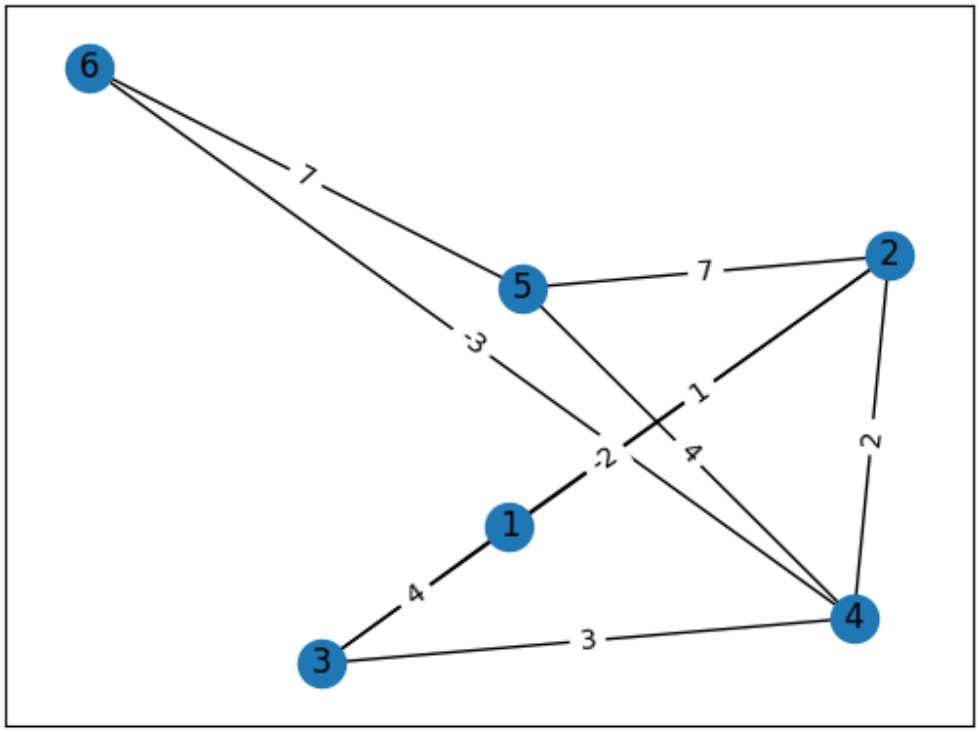

Рисунок 18 — Граф с отрицательным циклом

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были закреплены знаний по теме алгоритмов сортировки и графовых алгоритмов.