ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		М.Н. Шелест
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №2

БАЗОВЫЕ АЛГОРИТМЫ НА ГРАФАХ

по курсу: Построение и анализ графовых моделей

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4117		Б.Р.Зарипов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

1. Цель работы

Реализовать и проверить на тестовом примере базовый алгоритм на графе. Сравнить быстродействие реализованного алгоритма на специальных графах.

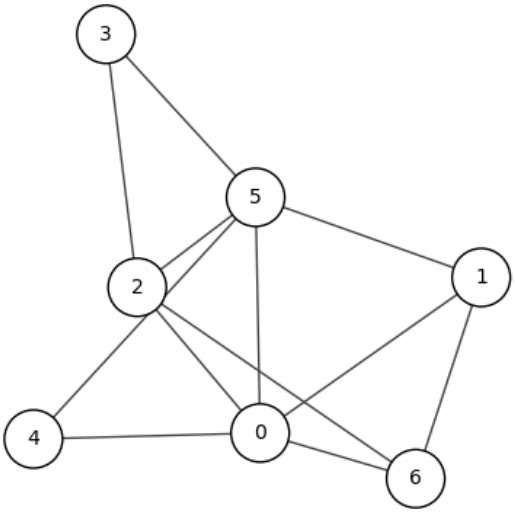
В рамках базового алгоритма необходимо реализовать «Нахождение метрики Жаккара между всеми парами вершин».

2. Изображение исходного графа

Листинг 1 Исходный код для отрисовки базового примера

```
g = igraph.Graph.Read_Edgelist('graph_example.txt', directed=False)
fig, ax = plt.subplots(figsize=(5,5))
igraph.plot(
    g,
    bbox = (300,300),
    target=ax,
    # layout="graphopt", # kamada_kawai
    vertex_label_color = 'black',
    vertex_label_size = 10,
    vertex_label = list(range(7)),
    vertex_size = 40,
    vertex_color = 'white',
    edge_arrow_width=7,
    edge_width=1,
    # edge_label=g.get_vertex_dataframe(),
    # edge_align_label=True,
    # edge_background='white'
)
plt.show()
```

Изображение 1 исходный граф



3. Описание метрики Жаккара

Метрика Жаккара — это статистическая мера сходства между двумя множествами вершин в графе. Она определяет коэффициент пересечения между двумя множествами вершин путем деления размера их общего пересечения на размер их объединения.

Для графа, метрика Жаккара может быть использована для оценки степени сходства между двумя группами вершин, например, для оценки сходства между сообществами в сети. Чем выше значение коэффициента Жаккара, тем больше сходство между двумя группами вершин.

Формула для расчета коэффициента Жаккара для двух множеств вершин A и B в графе выглядит следующим образом:

$$J(A,B) = |A \cap B| / |A \cup B|$$

Где $|A \cap B|$ обозначает размер общего пересечения между множествами соседних вершин A и B, а $|A \cup B|$ обозначает размер их объединения. Коэффициент Жаккара может принимать значения от 0 до 1, где 0 означает отсутствие сходства между множествами вершин, а 1 - полное сходство

4. Ручной расчет метрики Жаккара

Таблица 1 Ручной пересчет метрики Жаккара для исходного графа

	Число общих соседей						
	0	1	2	3	4	5	6
0	-1	2	2	2	1	3	2
1		-1	3	1	2	1	1
2			-1	1	2	2	1
3				-1	1	1	1
4					-1	1	1
5						-1	3
6							-1
	Длинна общего множества соседей						
	0	1	2	3	4	5	6
0	-1	6	7	5	6	7	6
1		-1	4	4	3	7	5
2			-1	5	4	7	6
3				-1	3	6	4
4					-1	6	4
5						-1	5
6							-1
	Метрика Жаккара						

	0	1	2	3	4	5	6
0		0,333333	0,285714	0,4	0,166667	0,428571	0,333333
1			0,75	0,25	0,666667	0,142857	0,2
2				0,2	0,5	0,285714	0,166667
3					0,333333	0,166667	0,25
4						0,166667	0,25
5							0,6
6							

5. Реализация алгоритма на языке программирования Python

Листинг 2 Получение метрики Жаккарта для всех возможных пар в графе

```
def create_jaccard_index_map(g: igraph.Graph) -> dict:
    jaccard_index = dict()  # метрика Жаккара
    for vertex_a in g.vs:  # перебор вершин
        for vertex_b in g.vs:
            if vertex_a == vertex_b:  # если вершины совпали
                continue
            neighbors_a = set(g.neighbors(vertex_a))  # множество соседей
            neighbors_b = set(g.neighbors(vertex_b))
            intersection = neighbors_a & neighbors_b  # пересечение
            union = neighbors_a | neighbors_b  # обьединение
            jaccard_index[(vertex_a.index, vertex_b.index)] = (
                len(intersection) / len(union),
                len(intersection),
                len(union)
            )
    return jaccard_index

create_jaccard_index_map(g)
```

Результат работы программы

```
{(0, 1): (0.3333333333333333, 2, 6),
 (0, 2): (0.2857142857142857, 2, 7),
 (0, 3): (0.4, 2, 5),
 (0, 4): (0.16666666666666666, 1, 6),
 (0, 5): (0.42857142857142855, 3, 7),
 (0, 6): (0.3333333333333333, 2, 6),
 (1, 0): (0.3333333333333333, 2, 6),
 (1, 2): (0.75, 3, 4),
 (1, 3): (0.25, 1, 4),
 (1, 4): (0.6666666666666666, 2, 3),
 (1, 5): (0.14285714285714285, 1, 7),
 (1, 6): (0.2, 1, 5),
 (2, 0): (0.2857142857142857, 2, 7),
 (2, 1): (0.75, 3, 4),
 (2, 3): (0.2, 1, 5),
 (2, 4): (0.5, 2, 4),
```

```
(2, 5): (0.2857142857142857, 2, 7),
(2, 6): (0.16666666666666666, 1, 6),
(3, 0): (0.4, 2, 5),
(3, 1): (0.25, 1, 4),
(3, 2): (0.2, 1, 5),
(3, 4): (0.3333333333333333, 1, 3),
(3, 5): (0.16666666666666666, 1, 6),
(3, 6): (0.25, 1, 4),
(4, 0): (0.16666666666666666, 1, 6),
(4, 1): (0.6666666666666666, 2, 3),
(4, 2): (0.5, 2, 4),
(4, 3): (0.3333333333333333, 1, 3),
(4, 5): (0.16666666666666666, 1, 6),
(4, 6): (0.25, 1, 4),
(5, 0): (0.42857142857142855, 3, 7),
(5, 1): (0.14285714285714285, 1, 7),
(5, 2): (0.2857142857142857, 2, 7),
(5, 3): (0.16666666666666666, 1, 6),
(5, 4): (0.16666666666666666, 1, 6),
(5, 6): (0.6, 3, 5),
(6, 0): (0.3333333333333333, 2, 6),
(6, 1): (0.2, 1, 5),
(6, 2): (0.16666666666666666, 1, 6),
(6, 3): (0.25, 1, 4),
(6, 4): (0.25, 1, 4),
(6, 5): (0.6, 3, 5)}
```

Где указаны

- Номера вершин
- Метрика Жаккарта для данной пары
- Длинна пересечения множеств соседей
- Длинна суммы множеств соседей

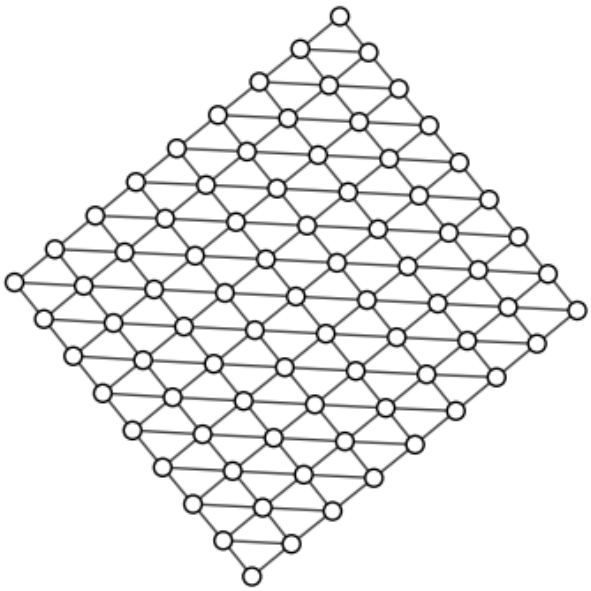
6. Правильная треугольная решетка

Листинг 3 Формирование правильной треугольной решетки

```python
def create_right_treeangle_sieve(n: int) -> Graph:
    # Создаем граф
    g = Graph()

    # Добавляем вершины
    num_vertices = n ** 2  # Общее количество вершин
    g.add_vertices(num_vertices)

    # Соединяем вершины внутри каждого треугольника
    for i in range(n):
        for j in range(n-1):
            g.add_edge(i*n+j, i*n+j+1)  # в строке

    for i in range(n-1):
        for j in range(n-1):
            g.add_edge(i*n+j, (i+1)*n+j+1)  # по диагонали вниз

    # Соединяем вершины соседних треугольников
```

```
    for i in range(n-1):
        for j in range(n):
            if (i+1)*n + j < num_vertices:
                g.add_edge(i*n+j, (i+1)*n+j)  # по вертикали
    return g

g = create_right_treeangle_sieve(9)
# Выводим граф
fig, ax = plt.subplots(figsize=(5,5))
igraph.plot(
    g,
    bbox = (300,300),
    target=ax,
    # layout="graphopt",  # kamada_kawai
    vertex_label_color = 'black',
    vertex_label_size = 10,
    vertex_size = 10,
    vertex_color = 'white',
    edge_arrow_width=7,
    edge_width=1,
    # edge_label=g.es["weight"],
    # edge_align_label=True,
    # edge_background='white'
)
plt.show()
```

Изображение 2 Правильная треугольная решетка



7. Проверка быстродействия

Листинг 4 Тестирование время работы от размера входных данных

```
from datetime import datetime


COUNT_OF_TESTS_PER_SIZE = 100
for size in range(5, 30):
    sieve = create_right_treeangle_sieve(size)
    start_time = datetime.now()
    for _ in range(COUNT_OF_TESTS_PER_SIZE):
        create_jaccard_index_map(sieve)
    end_time = datetime.now()
    runned_time = end_time - start_time
    print(size, runned_time.total_seconds())
```

Изображение 3 Время работы от размера решетки

8. Выводы

В ходе выполнения практической работы по изучению алгоритмов на графах были выполнены следующие этапы:

1. Был самостоятельно создан и визуализирован граф, содержащий 7 вершин. Граф был представлен в виде списка ребер для удобства работы с алгоритмами.

2. Построение метрики Жаккарта для всех пар вершин был успешно реализован с помощью средств ЭВМ. Это позволило протестировать алгоритм на созданном графе и получить результаты его работы.

3. Отладка работы алгоритма на созданном графе подтвердила корректность его функционирования и позволила убедиться в правильности результатов, полученных в ходе выполнения работы.

4. Была построена подпрограмма для построения правильной треугольной решетки.

5. Наконец, был построен график зависимости быстродействия выбранного алгоритма от количества узлов в правильной решетке. Этот график позволяет оценить производительность алгоритма в зависимости от изменения входных данных.

Таким образом, выполнение всех перечисленных этапов позволило не только изучить и реализовать базовые алгоритмы на графах, но и провести обширное исследование их работы, а также оценить их производительность. Полученные результаты могут быть полезны для дальнейшего изучения и развития алгоритмов на графах.