Дослідження роботи алгоритму Флойда-Воршала

Шевчук Іван, Шинкаренко Іван

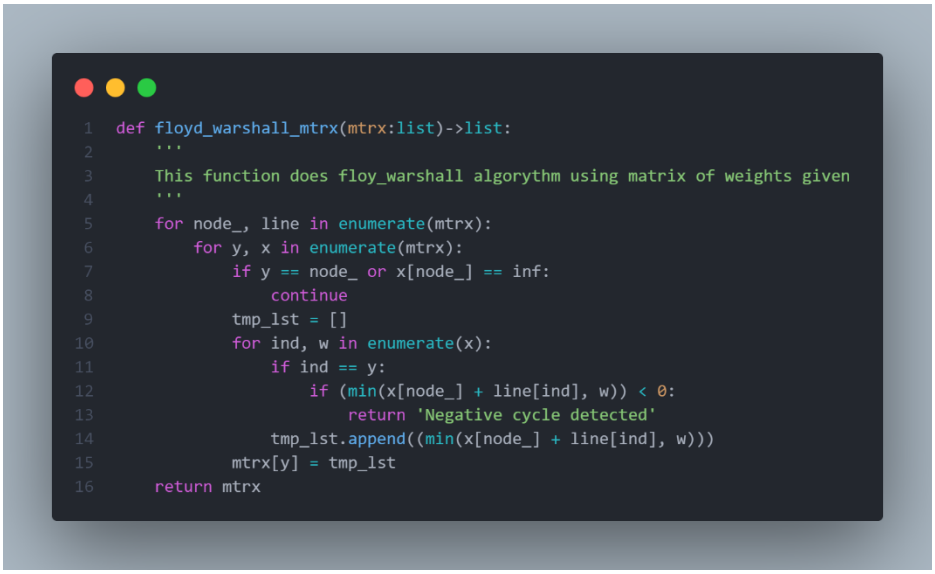
Нашою задачею було написання коду для алгоритму Флойда-Воршала та порівняння швидкості, ефективності та правильності роботи нашого алгоритму та вбудованого.

1. Наш Код

```python
def generate_mtrx(graph:nx.DiGraph)->list:
    '''
    This function generates a matrix of weights for given weighted graph
    '''
    weighted_mtrx = []
    for node_ in graph.nodes:
        tmp_lst = []
        for neighbor in graph.nodes:
            if neighbor == node_:
                tmp_lst.append(0)
            else:
                try:
                    tmp_lst.append(graph[node_][neighbor]['weight'])
                except KeyError:
                    tmp_lst.append(inf)
        weighted_mtrx.append(tmp_lst.copy())
    return weighted_mtrx
```

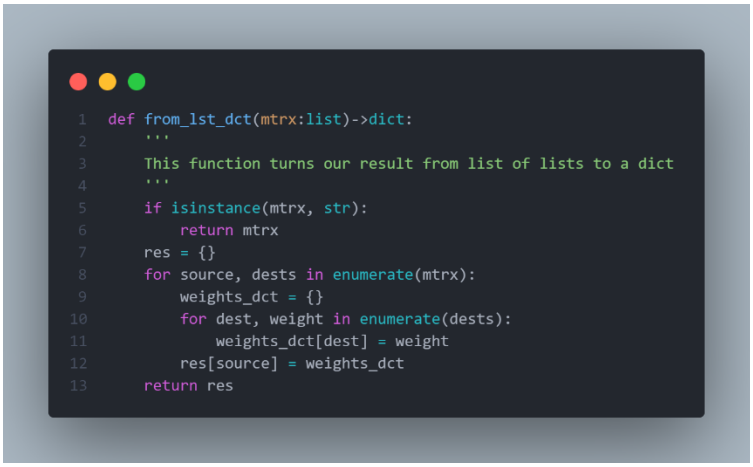
Виконання коду нашого алгоритму ми розділили на декілька функцій. Функція *generate_mtrx* на вхід бере граф створений за допомогою бібліотеки *networkx* та перетворює його у матрицю (список списків) ваг з якою може працювати алгоритм Флойда-Воршала. Матриця влаштована таким чином що номер рядку це номер вершини графа від якого йде шлях, а номер колонки це номер вершини до якої йде шлях. Якщо шляху між вершинами немає то вагою для такого шляху є безмежність.

Функція *floyd_warshall_mtrx.* На вхід дана функція бере список списків, матрицю, створену по принципу функції *generate_mtrx*, виконує алгоритм Флойда-Воршала на ній та повертає вже змінену матрицю ваг. Важливо помітити, що відбувається в рядках 11-13. Проходячись по значенням в матриці та обробляючи їх за алгоритмом, може виникнути ситуація коли шлях з вершини в неї саму стане



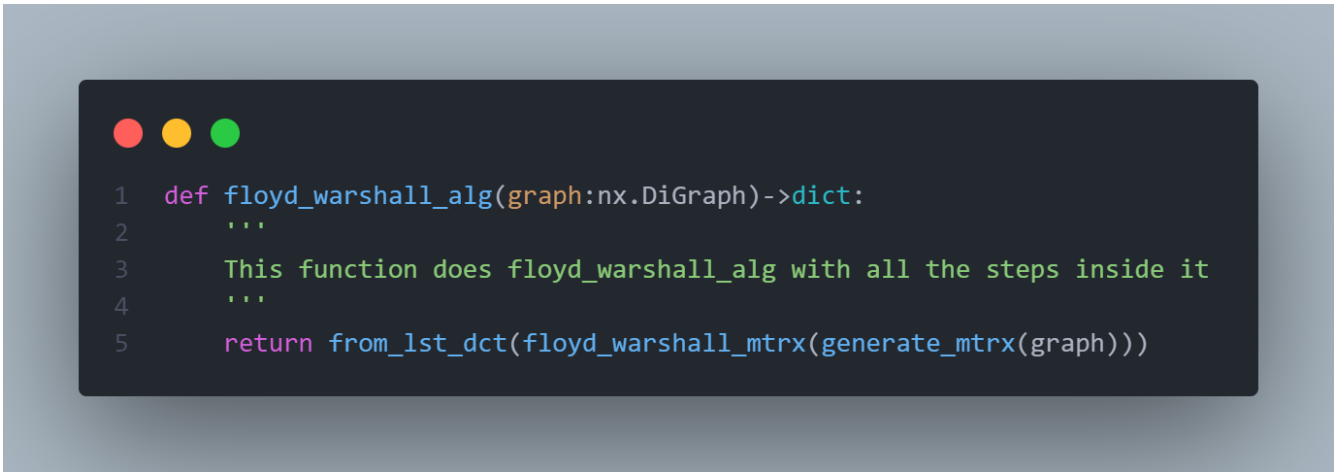
```python
def floyd_warshall_mtrx(mtrx:list)->list:
    '''
    This function does floy_warshall algorythm using matrix of weights given
    '''
    for node_, line in enumerate(mtrx):
        for y, x in enumerate(mtrx):
            if y == node_ or x[node_] == inf:
                continue
            tmp_lst = []
            for ind, w in enumerate(x):
                if ind == y:
                    if (min(x[node_] + line[ind], w)) < 0:
                        return 'Negative cycle detected'
                tmp_lst.append((min(x[node_] + line[ind], w)))
            mtrx[y] = tmp_lst
    return mtrx
```

від’ємним, це може свідчити лише про те що в даному графі присутній від’ємний цикл, тоді функція і повертає стрічку з повідомленням про знаходження негативного циклу.



```python
def from_lst_dct(mtrx:list)->dict:
    '''
    This function turns our result from list of lists to a dict
    '''
    if isinstance(mtrx, str):
        return mtrx
    res = {}
    for source, dests in enumerate(mtrx):
        weights_dct = {}
        for dest, weight in enumerate(dests):
            weights_dct[dest] = weight
        res[source] = weights_dct
    return res
```

Функція *from_lst_dct.* На вхід бере список списків, матрицю ваг, та повертає словник в якому ключем я вершина графу з якого ми починаємо шлях, а значенням є ще один словник ключем якого є номер вершини до якої шлях іде, а значенням є вага такого шляху. Якщо функція на вхід отримає стрічку (наприклад повідомлення про негативний цикл вона просто виведе її)



```python
def floyd_warshall_alg(graph:nx.DiGraph)->dict:
    '''
    This function does floyd_warshall_alg with all the steps inside it
    '''
    return from_lst_dct(floyd_warshall_mtrx(generate_mtrx(graph)))
```

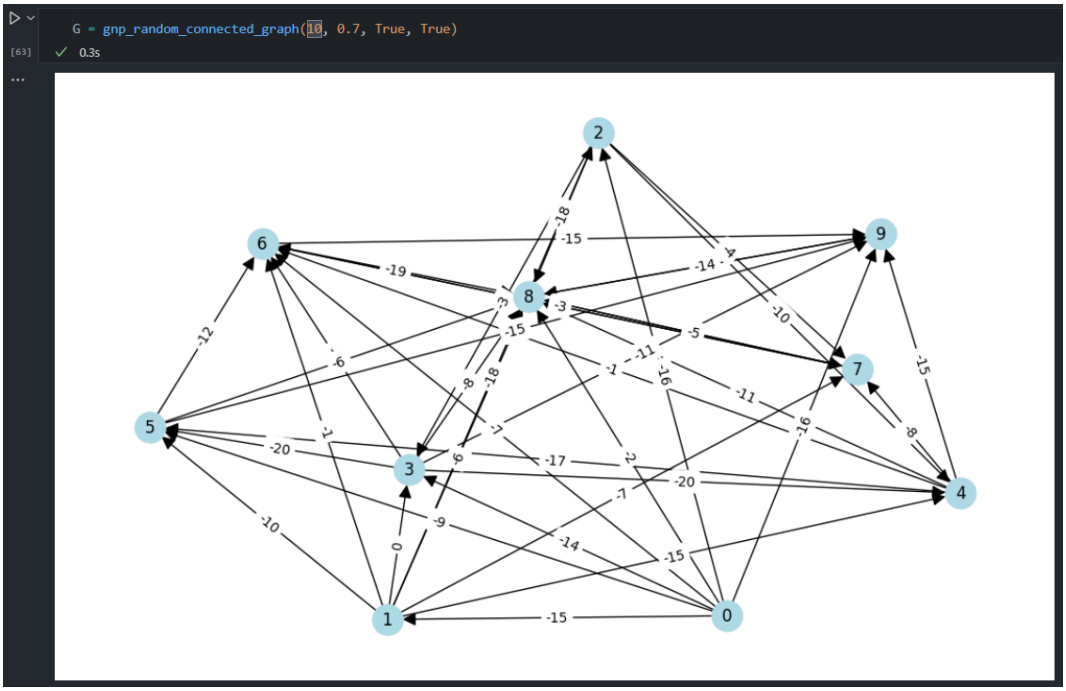
Фінальна функція *floyd_warshall_alg* яка використовує усі вище описані функції щоб вивести кінцевий результат, беручи на вхід граф створений за допомогою бібліотеки *network.*

2. Експеремент (порівняння)

Для зручнішого тестування було створено новий Jupiter notebook в якому знаходяться наші функції, функція для створення графу, та вбудована функція алгоритму Флойда-Форшала.

Одразу хочеться підмітити помилку у виконанні вбудованого алгоритму. Вбудований алгоритм погано визначає чи є в графі від’ємний цикл.

Для того щоб довести це твердження змінимо деякі значення у функції яка створює граф. Так, для визначення ваги шляху вона буде вибирати випадкове значення між -20 та 0. Спробуємо створити такий граф.



В цьому графі точно можна знайти від’ємний цикл (наприклад 6 -> 9 -> 8 -> 6). Але при запуску вбудованого алгоритму ми отримаємо звичайний, начебто правильний результат.

```
# pred is a dictionary of predecessors, dist is a dictionary of distances dictionaries
try:
    pred, dist = floyd_warshall_predecessor_and_distance(G)
    for k, v in dist.items():
        print(f"Distances with {k} source:", dict(v))
except:
    print("Negative cycle detected")
```

✓ 0.0s

```
Distances with 0 source: {0: 0, 9: -629, 1: -15, 2: -33, 3: -36, 5: -73, 6: -368, 8: -420, 4: -56, 7: -376}
Distances with 1 source: {1: 0, 3: -21, 2: -18, 4: -41, 5: -58, 6: -353, 7: -361, 8: -405, 0: inf, 9: -614}
Distances with 2 source: {2: 0, 4: -23, 3: -3, 7: -343, 8: -387, 0: inf, 1: inf, 5: -40, 6: -335, 9: -596}
Distances with 3 source: {3: 0, 9: -593, 4: -20, 5: -37, 6: -332, 8: -384, 0: inf, 1: inf, 2: inf, 7: -340}
Distances with 4 source: {4: 0, 8: -364, 5: -17, 6: -312, 7: -320, 9: -573, 0: inf, 1: inf, 2: inf, 3: inf}
Distances with 5 source: {5: 0, 9: -556, 6: -295, 8: -347, 0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 7: -303}
Distances with 6 source: {6: -283, 7: -291, 8: -335, 9: -544, 0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 5: inf}
Distances with 7 source: {7: -294, 6: -286, 8: -338, 0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 5: inf, 9: -547}
Distances with 8 source: {8: -365, 7: -321, 9: -574, 0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 5: inf, 6: -313}
Distances with 9 source: {9: -730, 8: -521, 0: inf, 1: inf, 2: inf, 3: inf, 4: inf, 5: inf, 6: -469, 7: -477}
```

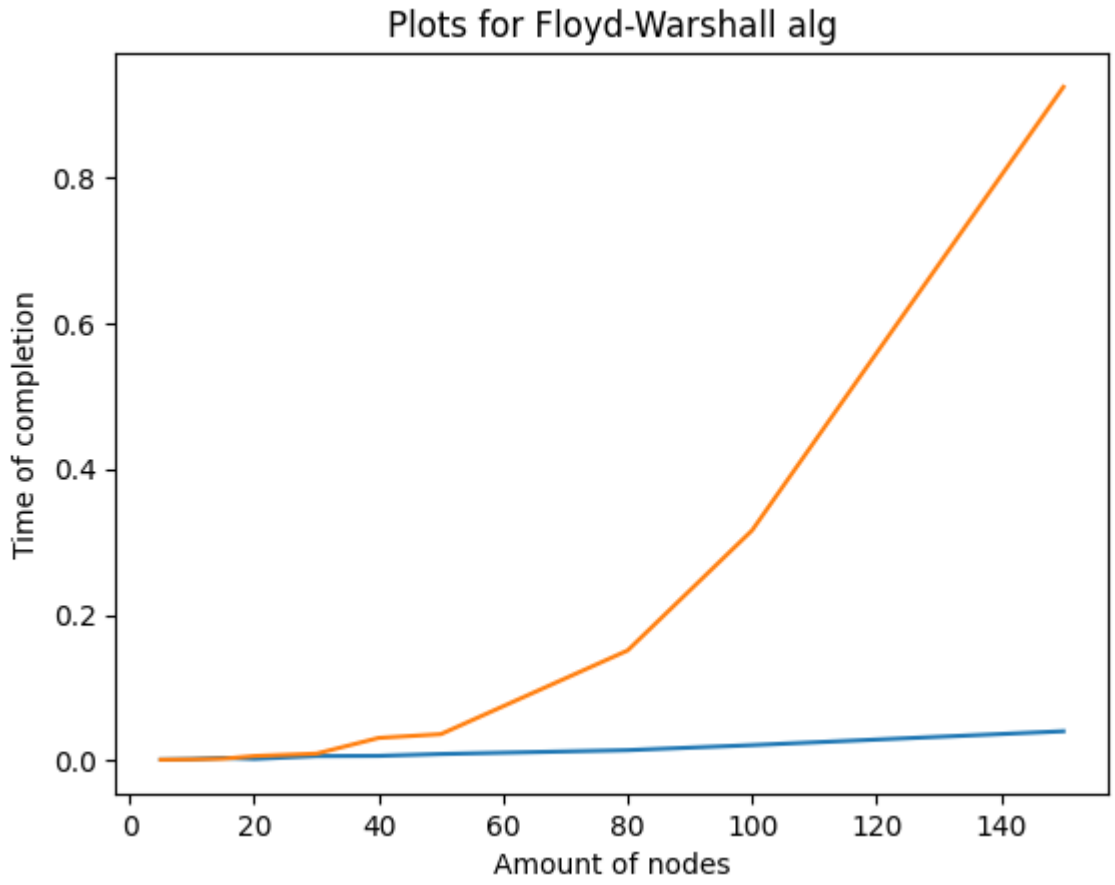
Наш алгоритм же знаходить від’ємний цикл та виводить відповідне повідомлення

```
res = floyd_warshall_alg(G)
try:
    for k, v in res.items():
        print(f"Distances with {k} source:", dict(v))
except:
    print(res)
```

[65] ✓ 0.0s

```
Negative cycle detected
```

Але повернемося до основного завдання.
Після запуску обидвох алгоритмів з графами які мають 5, 10, 15, 20, 30, 40, 50, 80, 100 та 150 ми отримали такий графік:



Синьою лінією позначено залежність кількості вершин від часу алгоритму написаного нами, а оранжевим вбудованого алгоритму. Може здатися що код написаний нами над швидкий, але річ у тім що у більшості випадків код написаний нами знаходив від’ємний цикл і як тільки це робив повертав повідомлення, коли як вбудований алгоритм працював і працював довго.і