Звіт до лабораторної роботи №1

Виконали: Дмитро Хамула, Лисун Тарас

Завдання лабораторної роботи: власноруч реалізувати алгоритми пошуку найкоротшого шляху в графі та пошуку мінімального каркасу

Посилання на репозиторій github: https://github.com/taraslysun/discrete_math_lab1

Завдання 1(Алгоритм Краскала):

Код алгоритму знаходиться у файлі prim_kruskal.py

Повний звіт по алгоритму знаходиться у файлі kruskal_report.ipynb

```python
def kruskal_algorithm(graph, algorithm=None) -> list[tuple[int, int, dict]]:
    '''
    Perform Kruskal's algorithm to find minimum panning tree
    Graph is given as list of edges with
    Returns list of tuples with information about minimum planning tree edges
    '''
    sorted_edges = sorted(list(graph.edges(data=True)), key=lambda x: x[2]['weight'])
    list_of_nodes = [set([node]) for node in list(graph.nodes())]
    res = []
    while len(list_of_nodes) > 1:
        for edge in sorted_edges:
            for node in list_of_nodes:
                if edge[0] in node and edge[1] not in node:
                    list_of_nodes.remove(node)
                    for node_set in list_of_nodes:
                        if edge[1] in node_set:
                            list_of_nodes.remove(node_set)
                            list_of_nodes.append(node.union(node_set))
                            break
                    res.append(edge)
                    break
    return res
```

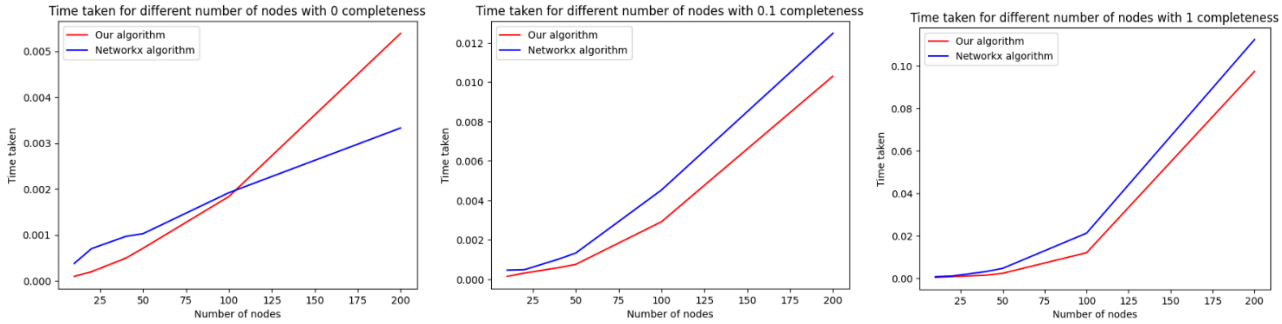
Функція приймає список кортежів-ребер графа. Та повертає мінімальний каркас у вигляді списку ребер. Алгоритм здебільшого реалізовано за псевдокодом.

Порівнювання швидкості роботи відбувалось із вбудованим в бібліотеку networkx.algorithms методом tree.minimum_spanning_tree.

За результатами досліджень наш алгоритм працює приблизно вдвічі швидше, ніж вбудований

```
Time taken for our algorithm on 10 nodes graph with 1 completeness:  0.0002624537944793701

100%|██████████| 1000/1000 [00:01<00:00, 960.96it/s]

Time taken for networkx algorithm on 10 nodes graph with 1 completeness:  0.0005517024993896484

100%|██████████| 1000/1000 [00:00<00:00, 1466.85it/s]
100%|██████████| 1000/1000 [00:00<00:00, 1901.72it/s]

Our algorithm works 204 % faster than networkx algorithm
```

Також є графіки швидкості виконання алгоритму на різній кількості вершин



Завдання 2(алгоритм Флойда-Воршала):

Код алгоритму знаходиться у файлі prim_kruskal.py

Повний звіт по алгоритму знаходиться у файлі floyd_report.ipynb

```python
def floyd_algorithm(graph: List, nodes: int) -> List:
    '''
    (List, int) -> List
    Performs Floyd's algorithm to find all pairs shortest path
    '''
    # Потрібні змінні
    inf = 10 ** 5

    # Пуста матриця ваг
    matrix = [[] * nodes] * nodes
    for a in range (0, nodes):
        matrix[a] = [inf] * nodes

    # Заповнена матриця ваг
    for t in graph:
        matrix[t[0]][t[1]] = t[2]['weight']

    for u in range (0, nodes):
        matrix[u][u] = 0

    # Аглоритм Флойда
    for k in range (0, nodes):
        for i in range (0, nodes):
            for j in range (0, nodes):
                matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j])

    for u in range (0, nodes):
        if matrix[u][u] < 0:
            print ('Negative cycle detected!')

            return False

    return matrix
```

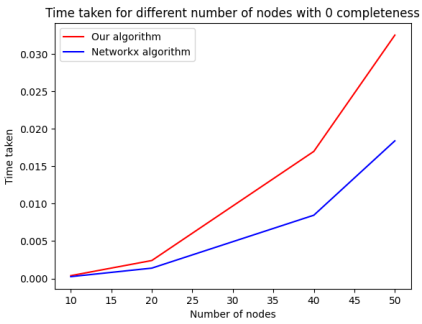
Функція приймає граф у вигляді списку ребер та кількість вершин графа

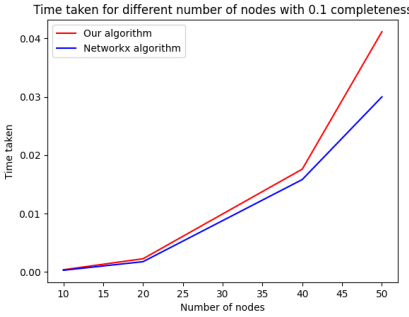
Повертає матрицю ваг

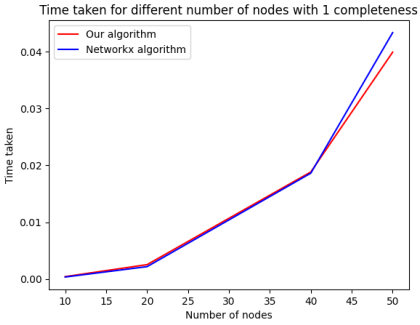
Здебільшого наш алгоритм працює на однаковій швидкості із вбудованим на малій кількості вершин, проте трохи повільніше, коли кількість вершин збільшується. Якщо граф є повним, то наш алгоритм працює трішки швидше.

```
100%|██████████| 1000/1000 [00:00<00:00, 3004.97it/s]

Time taken for our algorithm on 10 nodes graph with 1 completeness:  0.00032924461364746094

100%|██████████| 1000/1000 [00:00<00:00, 3190.79it/s]

Time taken for networkx algorithm on 10 nodes graph with 1 completeness:  0.0003116559982299805

100%|██████████| 1000/1000 [00:00<00:00, 2925.84it/s]
100%|██████████| 1000/1000 [00:00<00:00, 3064.38it/s]

Our algorithm works 104 % faster than networkx algorithm
```

Також є графіки порівнянь швидкості роботи нашого алгоритму із вбудованим







Завдання 3(Дерево рішень)

Код знаходиться у файлі tree.py. Файл train.py використовується для тренування нашого алгоритму.

Висновки:

Загалом під час виконання лабораторної роботи ми змогли поглибити свої знання у алгоритмах пошуку найкоротшого шляху та мінімального каркасу та закодити їх власноруч, а також зрозуміти роботу DecisionTreeClassifier та трішки познайомитись із машинним навчанням.