Звіт з дискретної математики

Лабораторна 1

Команда 3

Луцик Віталій та Остап Павлишин

Github repo - https://github.com/Vitalii-pr/lab_2.git

У цій практичній роботі ми написали 3 алгоритми:

Kruskal

Lustyk Vitalii

```python
class DisjointSet:
    def __init__(self, vertices):
        self.parent = {v: v for v in vertices}

    def find(self, vertex):
        if self.parent[vertex] != vertex:
            self.parent[vertex] = self.find(self.parent[vertex])
        return self.parent[vertex]

    def union(self, u, v):
        self.parent[self.find(u)] = self.find(v)


def kruskal(graph):
    vertices = {v for edge in graph for v in edge[:2]}  # Extracting unique vertices from edges
    res = []
    disjoint_set = DisjointSet(vertices)
    edges = sorted(graph, key=lambda x: x[2]['weight'])  # sort edges by weight

    for u, v, data in edges:
        if disjoint_set.find(u) != disjoint_set.find(v):
            res.append((u, v, {'weight': data['weight']}))
            disjoint_set.union(u, v)

    return res
```

Floyd-Worshall

Lustyk Vitalii

```python
INF = float('inf')

def adjacency_matrix(graph):
    num_of_nodes = len(graph.nodes())

    adj_matrix = [[INF] * num_of_nodes for _ in range(num_of_nodes)]

    for u, v, w in graph.edges(data=True):
        adj_matrix[u][v] = w.get('weight')
        if not graph.is_directed():
            adj_matrix[v][u] = w.get('weight')

    for i in range(num_of_nodes):
        adj_matrix[i][i] = 0

    return adj_matrix

def floyd_worshall(graph):
    res = dict()
    adj_matrix = adjacency_matrix(graph)
    num_of_nodes = len(adj_matrix)

    for k in range(num_of_nodes):
        for i in range(num_of_nodes):
            for j in range(num_of_nodes):
                adj_matrix[i][j] = min(adj_matrix[i][j], adj_matrix[i][k] + adj_matrix[k][j])

    for i in range(len(adj_matrix)):
        res[f'Distance with {i} source'] = {j:adj_matrix[i][j] for j in range(len(adj_matrix[i]))}

    return res
```

Bellman-Ford

Ostap Pavlyshun

```python
'''Bellman Ford'''

def bellman_ford(lst_edg_wei: list) -> None:
    '''
    Bellman Ford algorithm for finding the shortest distance between vertices
    '''
    lst = list(map(lambda x: (x[0], x[1], x[2]['weight']), list(lst_edg_wei.edges(data=True))))
    unique_vertices = set(list(map(lambda x: x[0], lst)) + list(map(lambda x: x[1], lst)))
    shortest_distance, predecessor = initialize_source(unique_vertices)
    for _ in range(len(lst) - 1):
        for v1, v2, w in lst:
            shortest_distance, predecessor = relax(v1, v2, w, shortest_distance, predecessor)
    for vertice1, vertice2, weight in lst:
        if shortest_distance[vertice2] > shortest_distance[vertice1] + weight:
            return "Negative cycle detected"

    new_dist = {vert:weight for vert, weight in shortest_distance.items() if weight != float('inf')}
    new_pred = {vert:old_lst for vert, old_lst in predecessor.items() if vert == 0 or old_lst != []}
    return new_dist, new_pred

def relax(v1, v2, weight, d, p):
    '''
    Relaxes given edge and updates distance dictionary
    and predecessor dictionary.
    '''
    if d[v2] > d[v1] + weight:
        d[v2] = d[v1] + weight
        p[v2] = [v1]
    return d, p

def initialize_source(vertices, start_v=0):
    '''
    Creates shortest_distance with inf edges for later updates
    and creates predecessor dictionary for later new previous vertices
    '''
    shortest_distance = {}
    predecessor = {}
    for v in vertices:
        shortest_distance[v] = float('inf')
        predecessor[v] = []
    shortest_distance[start_v] = 0
    return shortest_distance, predecessor
```

Також Остап написав друге задання Decision Tree Classifier.

Всі ці завдання ви зможете знайти на гітхабі за відповідними назвами, та в Graph Generation є всі алгоритми, та їхні графіки.

І також опис до алгоритмів.