Міні звіт команди якась там

Учасники: Депутат Антон і Головін Максим

```python
def prims_algorithm(G, start):
    prim_tree = []# our tree
    total_weight = 0
    edges = {our_node: (float('inf'), None) for our_node in G.nodes()}#storage for our weights for nodes are
    edges[start] = (0, None)
    visited = [start]
    new_node = start


    while len(visited) < len(G.nodes()):#visits all nodes
        for node in G.neighbors(new_node):
            if node not in visited:
                if G[new_node][node]['weight'] < edges[node][0]:#updates the shortest edge if found
                    edges[node] = (G[new_node][node]['weight'], new_node)

        our_edge_weight = float('inf')#founds the shortest edge that contains our node
        node_to_append = None
        for edge in edges:
            if edge not in visited:
                if edges[edge][0] < our_edge_weight:
                    our_edge_weight = edges[edge][0]
                    node_to_append = edge

        total_weight += our_edge_weight#updates everything if when found
        visited.append(node_to_append)
        prim_tree.append((new_node, node_to_append, our_edge_weight))
        new_node = node_to_append

    return prim_tree, total_weight#returns the tree
```

✓ 0.0s Python

```
([(0, 7, -4), (7, 9, -5), (9, 1, -4), (1, 5, -3), (5, 6, 0), (6, 4, -1), (4, 3, 1), (3, 8, -1), (8, 2, 7)], -10)
```

Перша частина це наша версія алгоритму прімаю вона бере найменше ребро і добавляє до одної з її вершин найменше аж до n-1 ітерації.

```python
def floyd(inp):
    """
    Floyd warshalls algorthm
    """
    twins = []
    lenth = list(inp.nodes())[-1]
    lst = list(inp.edges(data=True))
    for i in range(lenth + 1):#twins is a memory storage to which we will add all the lenth
        twins.append([])
        for g in range(lenth + 1):
            if g == i:
                twins[i].append([g, 0])# all i,i cells are 0
            else:
                twins[i].append([g])
    for i in lst:
        twins[i[0]][i[1]].append(i[2]["weight"])# append all edges
    for k in range(lenth + 1):#what floyd worshals algorithm does
        for i in range(lenth + 1):
            for j in range(lenth + 1):
                try:
                    if twins[i][j][1] > twins[i][k][1] + twins[k][j][1]:
                        twins[i][j][1] = twins[i][k][1] + twins[k][j][1]
                except IndexError:
                    try:
                        if twins[i][j][1]:
                            continue
                    except IndexError:
                        try:
                            twins[i][j].append(twins[i][k][1] + twins[k][j][1])
                        except IndexError:
                            continue
    return twins
```

Другою є алгоритм флойда воршала. Він спочатку ствоє ліст в який пхає всі ваги ребер які має граф. Опісля проходить по таблиці алгоритмом схожим до воршала рахуючи такі попарні обєднання кластерів ребер які будуть знижувати загальну вагу шляху. Так створюється таблиця оптимальних шляхів з кожної точки в кожну.

```
Distance with 0 source to 0: -2
Distance with 0 source to 1: 18
Distance with 0 source to 2: 2
Distance with 0 source to 3: 6
Distance with 0 source to 4: 20
Distance with 0 source to 5: 0
Distance with 0 source to 6: -8
Distance with 0 source to 7: 5
Distance with 0 source to 8: 1
Distance with 0 source to 9: 16
Distance with 1 source to 0: -9
Distance with 1 source to 1: 0
Distance with 1 source to 2: -5
Distance with 1 source to 3: -1
Distance with 1 source to 4: 4
Distance with 1 source to 5: -7
Distance with 1 source to 6: -15
Distance with 1 source to 7: -2
Distance with 1 source to 8: -6
Distance with 1 source to 9: 9
Distance with 2 source to 0: -6
Distance with 2 source to 1: 14
Distance with 2 source to 2: -2
Distance with 2 source to 3: 2
Distance with 2 source to 4: 16
...
Distance with 9 source to 6: -11
Distance with 9 source to 7: 2
Distance with 9 source to 8: -2
Distance with 9 source to 9: 0
```

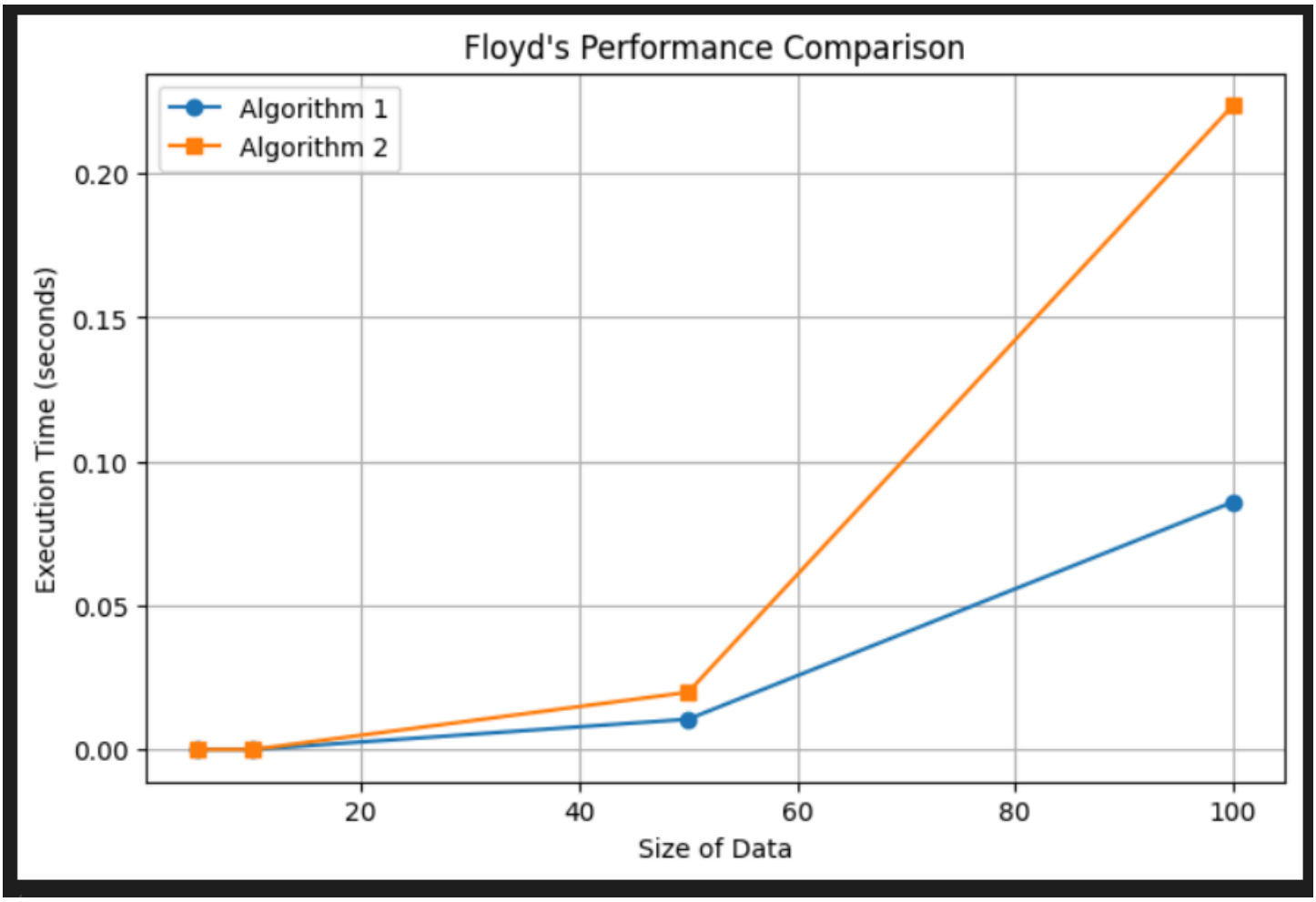
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Приклад виводу нашого алгоритму

```python
NUM_OF_ITERATIONS = 100
time_taken = 0
for i in tqdm(range(NUM_OF_ITERATIONS)):

    # note that we should not measure time of graph creation
    G = gnp_random_connected_graph(100, 0.4, False)

    start = time.time()
    floyd_warshall_predecessor_and_distance(G)
    end = time.time()

    time_taken += end - start

time_taken / NUM_OF_ITERATIONS

NUM_OF_ITERATIONS = 100
time_taken = 0
for i in tqdm(range(NUM_OF_ITERATIONS)):

    # note that we should not measure time of graph creation
    G = gnp_random_connected_graph(100, 0.4, False)

    start = time.time()
    floyd(G)
    end = time.time()

    time_taken += end - start

time_taken / NUM_OF_ITERATIONS
```

[66] ✓ 40.7s Python

```
100%|██████████| 100/100 [00:11<00:00,  8.36it/s]
100%|██████████| 100/100 [00:28<00:00,  3.48it/s]

0.2818081164360046
```



А це тестер по часу який порівнює алгоритм флойда з його нетворкх версією. Як можна побачити наш алгоритм в 2 рази менш ефективнийб але, якщо чесно, у нетворкх стаж більший ніж два джуна. Аналогічна версія є для алгоритму пріма

Далі іде трохи більший алгоритм, будівельник дерев.

```python
class Node:
    """
    :param X: numpy array of form [[feature1,feature2, ... featureN], ...] (i.e. [[1.5, 5.4, 3.2, 9.8] , ...] for case with iris d.s.)
    :param y: numpy array of from [class1, class2, ...] (i.e. [0,1,1,2,1,0,...] for case with iris d.s.)
    """
    def __init__(self, X, y):
        self.X = X
        self.y = y
        self.feature_index = None
        self.threshold = None
        self.left = None
        self.right = None
        self.label = self._most_common_label()

    def _most_common_label(self):
        values, counts = np.unique(self.y, return_counts=True)
        return values[np.argmax(counts)]
```

[69] ✓ 0.0s Python

+ Code + Markdown

```python
class DecisionTreeClassifier:
    def __init__(self, max_depth=3):
        self.max_depth = max_depth
        self.tree = None
        self.number_of_classes = None

    def build_tree(self, X, y, depth):
        """
        Every man needs to plant a son, raise a house and build a tree
        """
        if depth < self.max_depth and len(np.unique(y)) > 1:
            best_feature, best_threshold = self.best_split(X, y)#finds the best feature and threshold for splitting
            if best_feature is not None:#splits the data into left and right child nodes
                left part = [
```

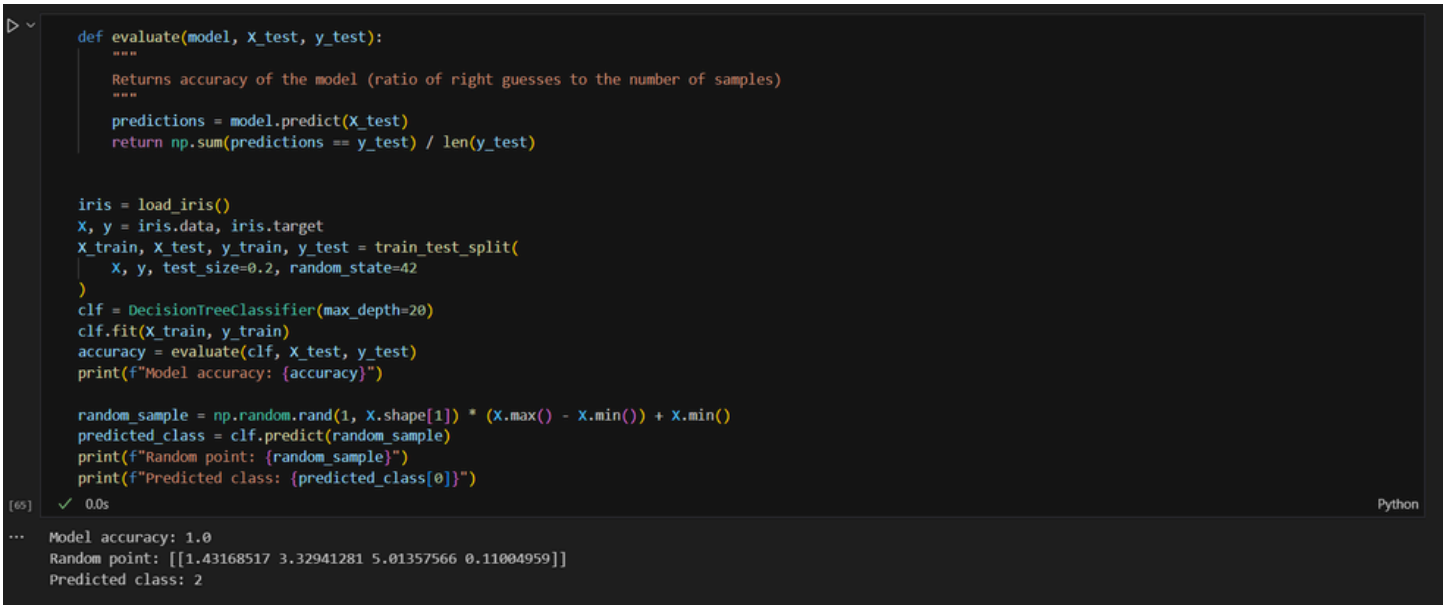
[70]

```
                ]
                right_part = [
                    i for i in range(len(X)) if X[i][best_feature] > best_threshold
                ]
                left_tree = self.build_tree(X[left_part], y[left_part], depth + 1)#creates a new node with the split feature and threshold
                right_tree = self.build_tree(X[right_part], y[right_part], depth + 1)
                node = Node(X, y)
                node.feature_index = best_feature
                node.threshold = best_threshold
                node.left = left_tree
                node.right = right_tree
                return node
        else:
            return Node(X, y)

    def best_split(self, X, y):
        """
        finds out the best split
        """
        max_info_gain = -float("inf")
        best_feature = None
        best_treshold = None
        for feature in range(X.shape[1]):#iterates over all features
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:#splits the data based on the threshold
                left_part = [i for i in range(len(X)) if X[i][feature] <= threshold]
                right_part = [i for i in range(len(X)) if X[i][feature] > threshold]
                if len(left_part) > 0 and len(right_part) > 0:#finds information gain and update the best split
                    info_gain = self.information_gain(y, y[left_part], y[right_part])
                    if info_gain > max_info_gain:
                        max_info_gain = info_gain
                        best_feature = feature
                        best_treshold = threshold
        return best_feature, best_treshold
```

```
    def information_gain(self, parent, left, right):
        """
        information gained
        """
        info = (
            self.gini_index(parent)
            - ((len(left) / len(parent)) * self.gini_index(left))
            - ((len(right) / len(parent)) * self.gini_index(right))
        )
        return info

    def gini_index(self, y):
        """
        gini impurity index
        """
        unique_classes = set(y)
        unique_classes_dict = {}
        for unique_class in unique_classes:#counts occurrences of each class
            count = 0
            for element in y:
                if element == unique_class:
                    count += 1
            unique_classes_dict[unique_class] = count
        gini = 1
        for j in unique_classes_dict.values():
            gini -= (j / len(y)) ** 2
        return gini
```

```
    def fit(self, X: npt.NDArray, y: npt.NDArray) -> None:
        """
        Basically, function that performs all the training (building of a tree)
        We recommend to use it as a wrapper of recursive building function
        """
        self.number_of_classes = np.unique(y).size
        self.tree = self.build_tree(X, y, depth=0)

    def predict(self, X):
        """
        Traverse the tree while there is a child
        and return the predicted class for it
        """
        return np.array([self._traverse_tree(self.tree, x) for x in X])

    def _traverse_tree(self, node, x):
        """
        recursively traverses the tree for prediction
        """
        if node.left is None and node.right is None:#goes through every node if it is a leaf stops else goes to the lef and right nodes respectively
            return node.label
        if node.feature_index is None:
            return node.label
        if x[node.feature_index] <= node.threshold:
            return self._traverse_tree(node.left, x)
        else:
            return self._traverse_tree(node.right, x)
```

[55] ✓ 0.0s Python



```python
def evaluate(model, X_test, y_test):
    """
    Returns accuracy of the model (ratio of right guesses to the number of samples)
    """
    predictions = model.predict(X_test)
    return np.sum(predictions == y_test) / len(y_test)


iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
clf = DecisionTreeClassifier(max_depth=20)
clf.fit(X_train, y_train)
accuracy = evaluate(clf, X_test, y_test)
print(f"Model accuracy: {accuracy}")

random_sample = np.random.rand(1, X.shape[1]) * (X.max() - X.min()) + X.min()
predicted_class = clf.predict(random_sample)
print(f"Random point: {random_sample}")
print(f"Predicted class: {predicted_class[0]}")
```

[65] ✓ 0.0s Python

```
Model accuracy: 1.0
Random point: [[1.43168517 3.32941281 5.01357566 0.11004959]]
Predicted class: 2
```

Наш алгоритм побудови дерева розділяється на чотири частини: Кожен віддільний нод, Обрахування найкращого спліта, побудова дерева і тестер. Для кожного нода є дуже багато властивостей основна з яких що він має два сини в які іде рукрсивно алгоритм побудови дерева. Алгоритм побудови дерева проходить рекурсивно в пошуках найкращого розрізу по кожному ноду дерева. ЯКщо один з синів листок він туди не заходить, а коли обидва то повертається назад. Розріз шукається за допомогою вибору рандомною точкою кожного алгоритму алгоритму джині який прораховується на кожному кроці на батьку і синах, а потім розраховуючи інформацію здобуту за цей розріз вирішує чи він найкращий. Тестер просто перевіряє чи програма створює ефективне дерево а далі створює рандомну точку якій визначається клас.

дякую за те що читали цей звіт