**Міністерство освіти і науки України**
**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**
**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

# Звіт

з лабораторної роботи № 4 з дисципліни
«ПІІС»

**Виконав(ла)** *ІТ-02 Макаров И.С.*
(шифр, прізвище, ім'я, по батькові)

**Перевірив** _____ (прізвище, ім'я, по батькові)

Київ 2021

**ЗМІСТ**

## МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Дослідження алгоритмів пошуку підрядочків у рядку та найкоротшого шляху та графі

## ЗАВДАННЯ

1. Реалізувати алгоритм ***Карпа-Рабіна***, обравши хеш-функцією результат ділення за модулем на номер Вашого варіанту. Протестувати роботу алгоритму над шаблонами у 6-8 символів.
2. Реалізувати алгоритм ***Дейкстри*** згідно заданого варіанту.
3. Реалізувати алгоритм ***Пріма*** згідно заданого варіанту.

**Варіант №10.**

1.1. Задано орієнтований граф та час переходу від однієї вершини до іншої: 27=3, 24=1, 31=3, 36=2, 35=2, 42=3, 47=3, 56=5, 57=2, 58=5, 62=1, 61=3, 65=2, 67=1, 68=3, 76=1, 78=2, 82=2, 84=2, 87=4. Необхідно знайти найкоротші відстані від заданої вершини до інших.

1.2. Задано неорієнтований граф та час переходу від однієї вершини до іншої: 23=3, 24=1, 31=3, 36=2, 35=2, 42=3, 47=3, 56=5, 57=2, 58=5, 62=1, 63=3, 65=2, 67=1, 68=3, 76=1, 78=2, 83=2, 84=2, 87=4. Необхідно побудувати мінімальне остовне дерево за допомогою алгоритму Пріма.

# ВИКОНАННЯ

**Вихідний код всіх алгоритмів**

**Rabin-Karp:**

https://github.com/kinfi4/AlgorithmsDataStructures/blob/main/Algorithms/Rabin-Karp%20Algorithm/algorithm.py

**Dijkstra:**

https://github.com/kinfi4/AlgorithmsDataStructures/blob/main/Algorithms/Dijkstra%E2%80%99s%20algorithm%20python/algorithm.py

**Prims:**

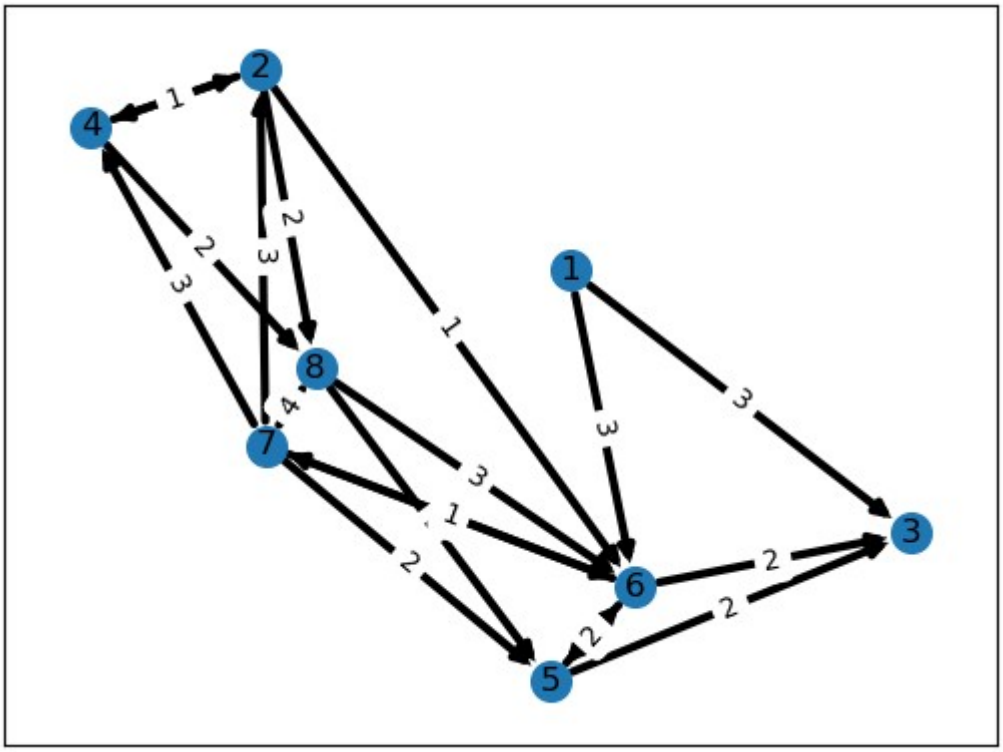
https://github.com/kinfi4/AlgorithmsDataStructures/blob/main/Algorithms/Prims%20Algorithm/algorithm.py

## ДЕЙКСТРА

```python
def get_linked_nodes(start_node_index, graph_matrix):
    """ Finds all the nodes that are linked to the start_node_index """
    for node_i, weight in enumerate(graph_matrix[start_node_index]):
        if weight > 0:
            yield node_i

kinfi4
def arg_min(weights, seen):
    """ Finds the lowest value of weight for node which is not in seen """
    min_index = -1
    min_value = max(weights)

    for index, weight in enumerate(weights):
        if weight < min_value and index not in seen:
            min_value = weight
            min_index = index

    return min_index

kinfi4
def dijkstra(graph_matrix, start_node):
    """ Gets the graph matrix and index of start node, and returns the list of weights """
    weights = [math.inf] * len(graph_matrix)
    weights[start_node] = 0
    seen = {start_node}

    while start_node != -1:
        for neighbor in get_linked_nodes(start_node, graph_matrix):
            if neighbor not in seen:
                neighbor_weight = weights[start_node] + graph_matrix[start_node][neighbor]
                if neighbor_weight < weights[neighbor]:
                    weights[neighbor] = neighbor_weight

        start_node = arg_min(weights, seen)
        if start_node > 0:
            seen.add(start_node)

    return weights
```

Якщо вам щось погано видно, то можете зайти просто на github, вибачте за незручності, я просто не знаю, як би його нормально заскрінить

**РЕЗУЛЬТАТ**




```
Index: 1 - distance: inf
Index: 2 - distance: 0
Index: 3 - distance: 3
Index: 4 - distance: 3
Index: 5 - distance: 4
Index: 6 - distance: 1
Index: 7 - distance: 2
Index: 8 - distance: 2
```

**Тут ми шукаємо відстань від ноди з Index = 2, до всіх інших.**

**АЛГОРИТМ ПРІМА**

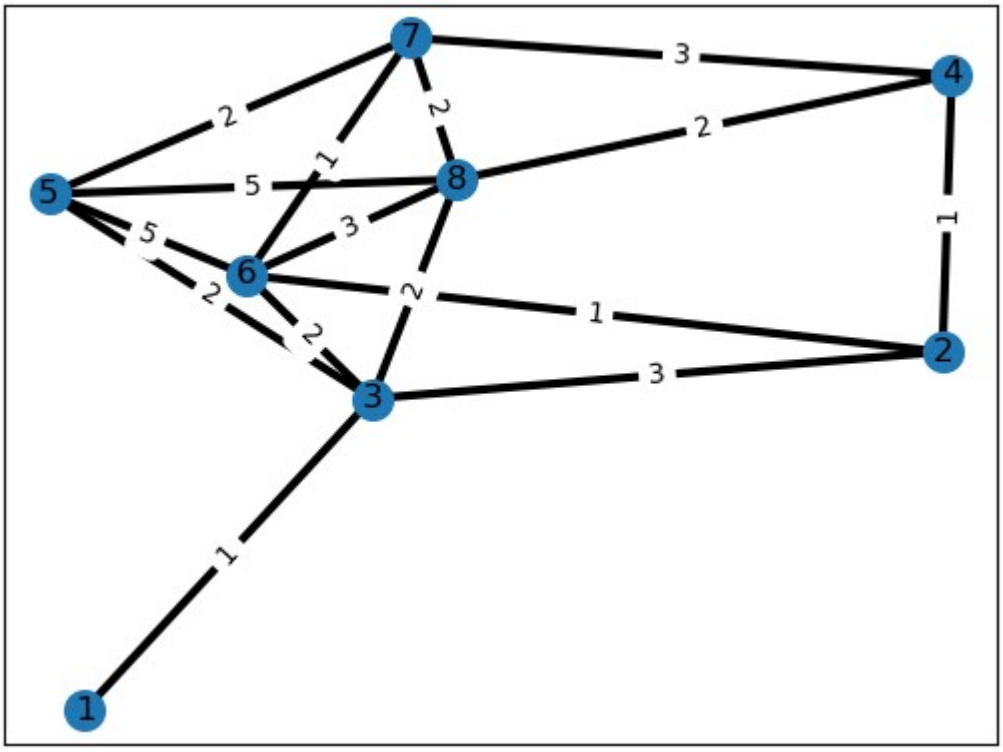
```
kinfi4
def search_closest_node(graph_matrix, visited):
    min_weight = max(row[i] for row in graph_matrix for i in range(len(row)))
    min_index = -1

    for visited_index in visited:
        for index, weight in enumerate(graph_matrix[visited_index]):
            if 0 < weight < min_weight and index not in visited:
                min_weight = weight
                min_index = index

    return [min_weight, min_index]


kinfi4
def prims_algorithm(graph_matrix: List[List[int]], starting_node: int):
    nodes = [(starting_node + 1, 0)]
    visited = {starting_node}
    to_visit = [i for i in range(len(graph_matrix)) if i != starting_node]

    for node in to_visit:
        weight, index = search_closest_node(graph_matrix, visited)
        nodes.append((index + 1, weight))
        visited.add(index)

    return nodes
```

Тут, будуючи мінімальне остовне дерево, я просто вказую список пар: вершина, відстань до цієї вершини, від її предка.

**РЕЗУЛЬТАТ**

```
/home/kinfi4/anaconda3/bin/python /home/kinfi4/AlgorithmsDataStructures
[(1, 0), (3, 1), (5, 2), (6, 2), (2, 1), (4, 1), (7, 1), (8, 2)]

Process finished with exit code 0
```

Починаємо ми тут з ноди з індексом 1

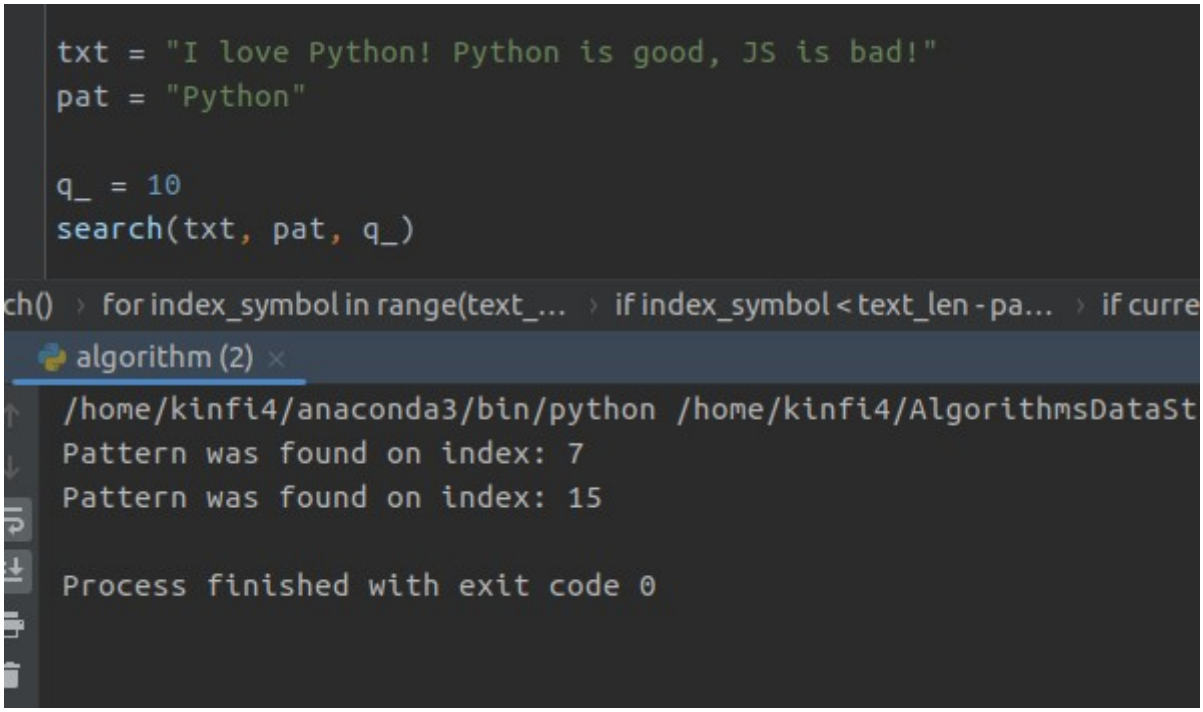



## РАБІНА КАРПА

```python
kinfi4
def get_hash(text: str, q: int) -> int:
    result = 0

    for i in range(len(text)):
        result = (B * result + ord(text[i])) % q

    return result


kinfi4
def search(text: str, pattern: str, q: int):
    pattern_len = len(pattern)
    text_len = len(text)

    if pattern_len > text_len:
        raise ValueError('Invalid pattern or text were specified. Pattern length can not be higher than text length')

    multiplier = 1
    for i in range(1, pattern_len):
        multiplier = (multiplier * B) % q

    pattern_hash = get_hash(pattern, q)
    current_window_hash = get_hash(text[:pattern_len], q)

    for index_symbol in range(text_len - pattern_len + 1):
        if pattern_hash == current_window_hash and text[index_symbol: index_symbol + pattern_len] == pattern:
            print(f'Pattern was found on index: {index_symbol}')

        if index_symbol < text_len - pattern_len:
            current_window_hash = (
                (current_window_hash - ord(text[index_symbol]) * multiplier)
                * B
                + ord(text[index_symbol + pattern_len])
            ) % q

            if current_window_hash < 0:
                current_window_hash += q
```

**В ньому я просто виводжу в консоль всі індекси по яким паттерн входить в строку.**

**РЕЗУЛЬТАТ**




**Висновок**: в лабораторній я познайомився ще з Зма новими алгоритмами, досить цікаво було.