Лабораторна робота №1

Виконали: Омельчук Олеся та Кічігіна Ангеліна

Завдання: реалізувати алгоритм Прима та Крускала для знаходження кістякового дерева мінімальної ваги. Порівняти ефективність роботи цих алгоритмів (перевірити, чи вона залежить від кількості ребер і заповненості графа)

Специфікація комп'ютера:

- Кількість ядер - 2
- Тактова частота - 2.7 GHz
- Пам'ять - 16 GiB
- Операційна система - Manjaro Linux

Алгоритм Крускала

```python
def kruskal(graph_and_weight: tuple) -> list[tuple]:
    """
    Get minimum spanning tree using Kruskal's algorithm.
    Returns: list of nodes.
    """
    carcass_edges = []

    G = graph_and_weight[0]
    edges = list(G.edges())
    num_of_nodes = len(G.nodes())
    weights = graph_and_weight[1]

    nodes = [{i} for i in range(num_of_nodes)]
    weights_edges = {}
    for i in range(len(weights)):
        weights_edges[edges[i]] = weights[i]
    weights_edges_sorted = sorted(weights_edges.items(), key=lambda x: x[1])

    for edge in weights_edges_sorted:
        node_1 = edge[0][0]
        node_2 = edge[0][1]
        sets_to_union = []
        for group in nodes:
            if node_1 in group and node_2 in group:
                break
            elif node_1 in group or node_2 in group:
                sets_to_union.append(group)

        if sets_to_union:
            carcass_edges.append(edge[0])
            nodes.append(sets_to_union[0].union(sets_to_union[1]))
            nodes.remove(sets_to_union[0])
            nodes.remove(sets_to_union[1])

    return carcass_edges
```

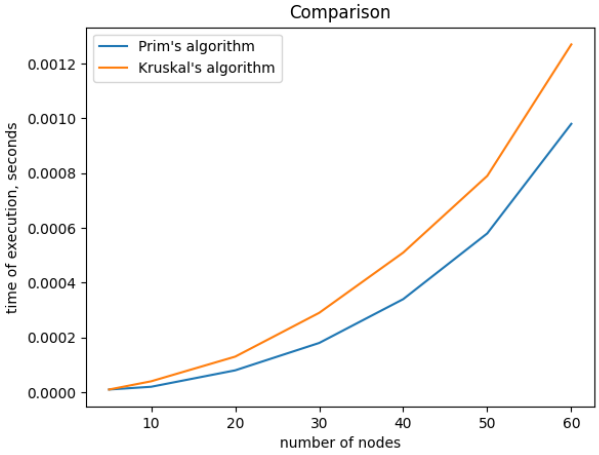
Алгоритм Прима

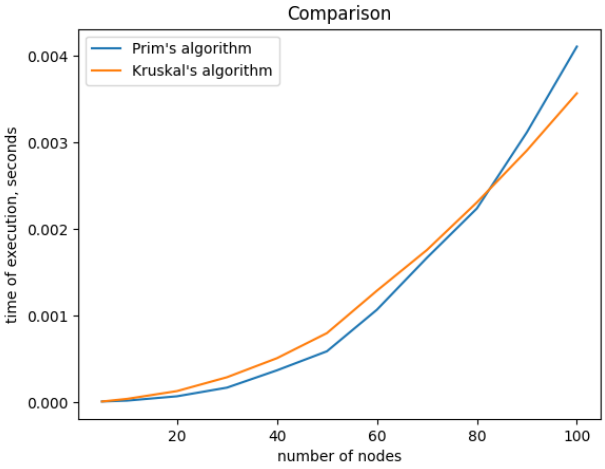
```python
def prim(graph_info: tuple) -> list[tuple]:
    """
    Get minimum spanning tree using Prim's algorithm.
    Returns: list of nodes.
    """
    minimum_spanning_tree = []

    G = graph_info[0]
    edges = list(G.edges())
    nodes_unvisited = list(G.nodes())
    weight_list = graph_info[1]
    graph = sorted(list(zip(edges, weight_list)), key=lambda x: x[1])

    nodes_visited = [nodes_unvisited[0]]
    del nodes_unvisited[0]

    while nodes_unvisited:
        for edge_weight in graph:
            edge = edge_weight[0]
            node_1 = edge[0]
            node_2 = edge[1]

            if node_1 in nodes_visited and node_2 in nodes_unvisited:
                minimum_spanning_tree.append(edge)
                graph.remove(edge_weight)
                nodes_visited.append(node_2)
                nodes_unvisited.remove(node_2)
                break

            elif node_2 in nodes_visited and node_1 in nodes_unvisited:
                minimum_spanning_tree.append(edge)
                graph.remove(edge_weight)
                nodes_visited.append(node_1)
                nodes_unvisited.remove(node_1)
                break

    return minimum_spanning_tree
```

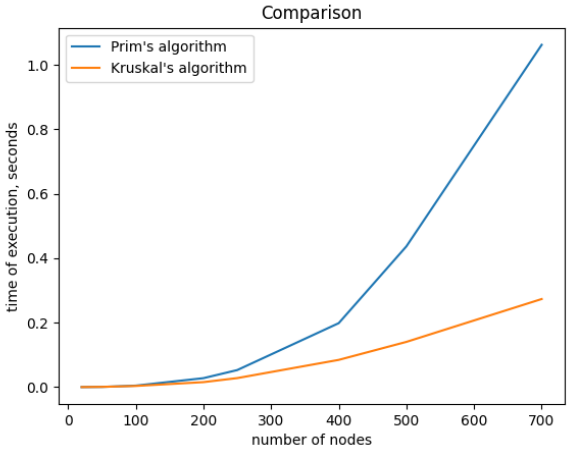
Код для проведення експериментів

```
num_of_nodes = [20, 50, 100, 200, 250, 400, 500, 700]
iterations_num = 15
completeness = 1

prim_time_execution = []
kruskal_time_execution = []

for nodes in num_of_nodes:
    graph = gnp_random_connected_graph(nodes, completeness)

    prim_all = []
    for i in range(iterations_num):
        start = time.time()
        prim(graph)
        end = time.time()
        time_taken = end-start
        prim_all.append(time_taken)
    prim_min_time = min(prim_all)
    prim_time_execution.append(round(prim_min_time, 5))

    kruskal_all = []
    for i in range(iterations_num):
        start = time.time()
        kruskal(graph)
        end = time.time()
        time_taken = end-start
        kruskal_all.append(time_taken)
    kruskal_min_time = min(kruskal_all)
    kruskal_time_execution.append(round(kruskal_min_time, 5))

plt.plot(num_of_nodes, prim_time_execution, label="Prim's algorithm")
plt.plot(num_of_nodes, kruskal_time_execution, label="Kruskal's algorithm")

plt.xlabel('number of nodes')
plt.ylabel('time of execution, seconds')
plt.title('Comparison')
plt.legend()

plt.show()
```

Експерименти

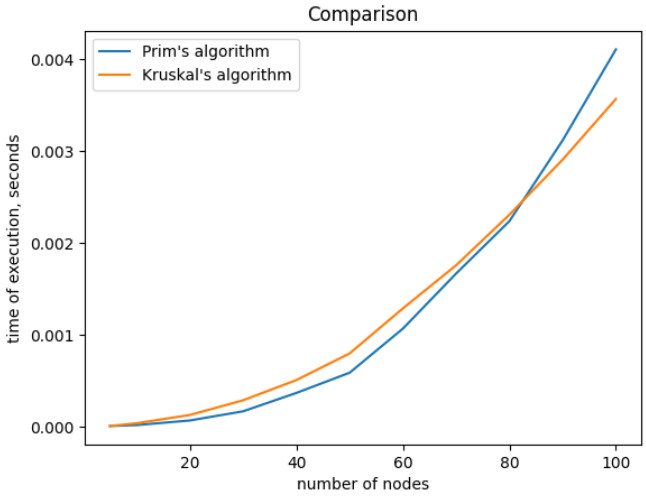




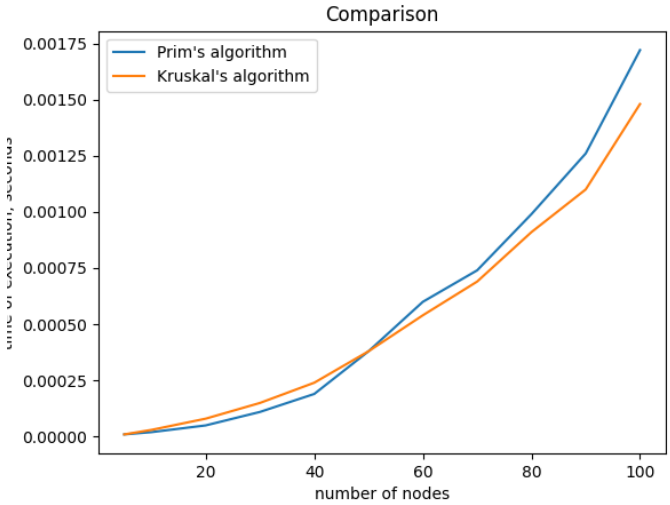


Дані графіки ілюструють залежність ефективності алгоритмів від **кількості ребер.** *(*при цьому completeness = 1)*

Як бачимо, для менших графів алгоритм Прима працює краще, проте коли ребер стає більше, алгоритм Крускала є значно швидшим.



Completeness = 1

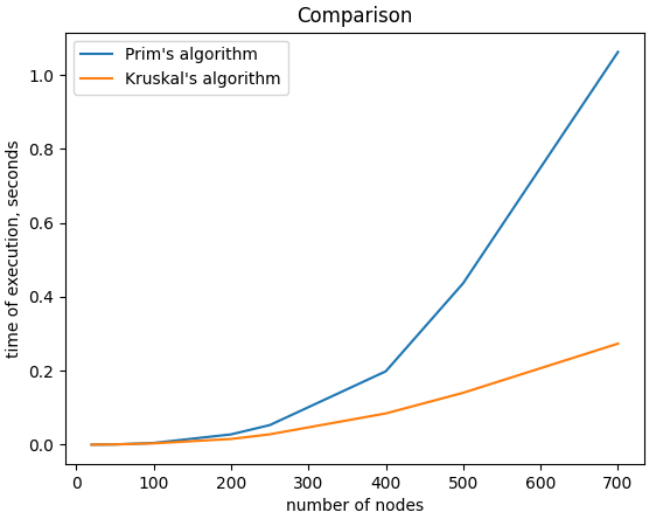


Completeness = 0.3

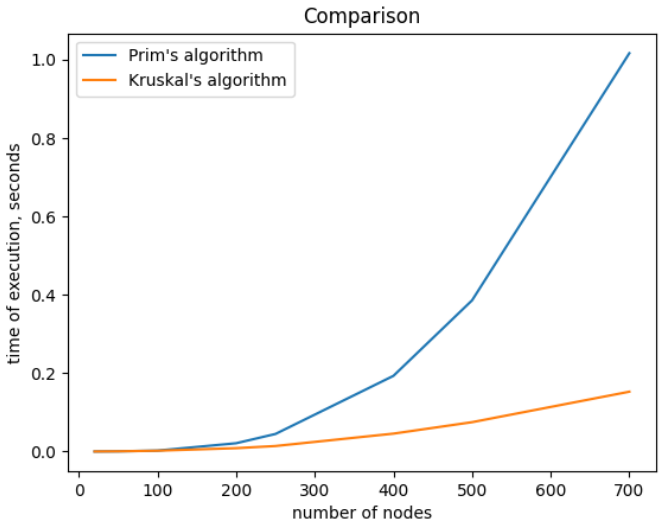
Тут ми експериментували із **заповненістю** графа.

Дивлячись на графіки, можна зробити висновок, що чим менша наповненість графа, тим швидше алгоритм Крускала «переганяє» Прима.

Проте, на великих графах заповненість особливо не грає ролі.



Completeness = 1



Completeness = 0.5

Висновок:

Отже, для повних графів з великою кількістю ребер (починаючи приблизно з 80-ти) краще використовувати алгоритм Крускала. Натомість, якщо повний граф має невелику кількість ребер, то алгоритм Прима працює швидше.

Якщо ж це неповний граф, то найкращим рішенням буде алгоритм Крускала, адже на великих графах його ефективність значно більша, а на малих різниця між швидкостями роботи двох алгоритмів дуже маленька.