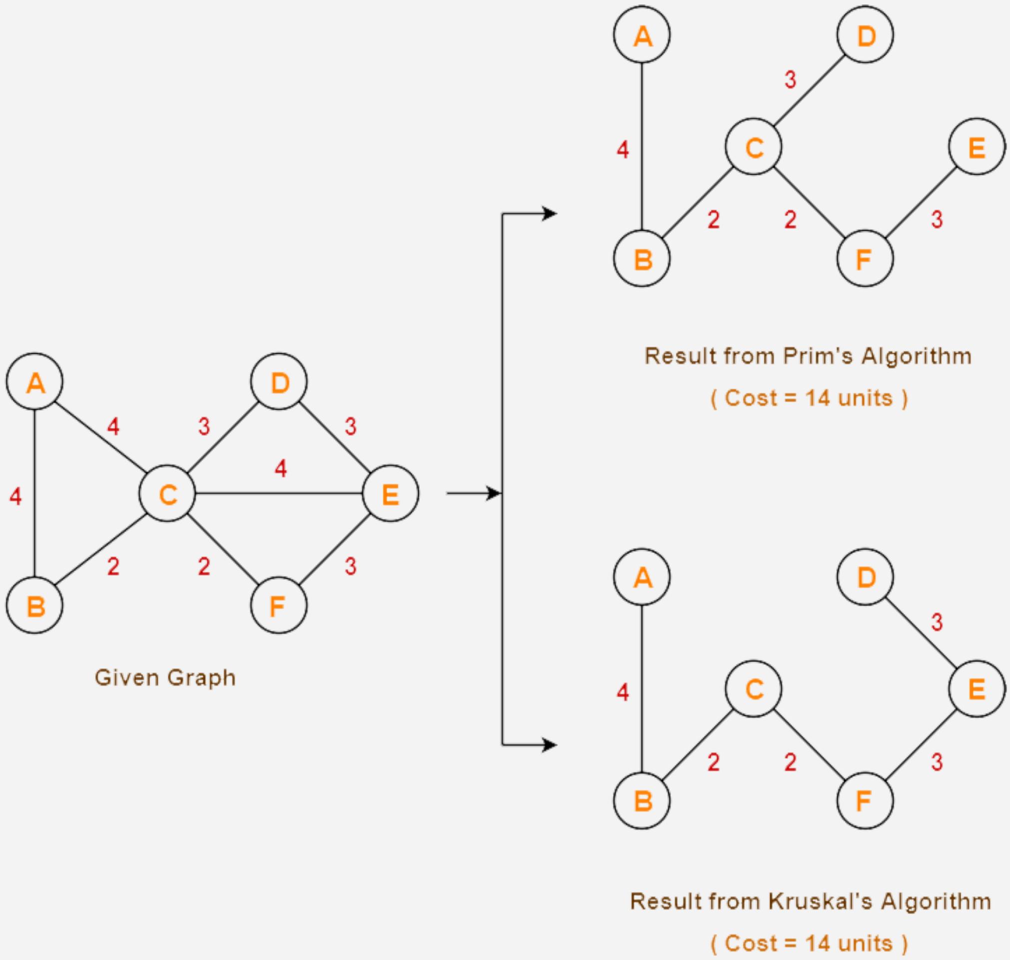
Звіт до лабораторної роботи з дискретної математики

Порівняння ефективності роботи алгоритмів Прима та Краскала

20.02.2022



Виконали студенти УКУ, першого курсу спеціальності КН

Мацишин Сергій, Шуляр Владислав, Конопада Олексій

➢ Алгоритм Прима

Даний алгоритм є жадібним і його використовують для побудови мінімального кістякового дерева зваженого неорієнтованого графу.

Покроково:

1. Обираємо довільну початкову вершину
2. Створюємо множину необраних вершин
3. Кожного разу обираємо ребро мінімальної ваги, яке належить до множини необраних вершин
4. Додаємо нове ребро до нашого дерева
5. Очищаємо нашу множину необраних вершин, забираємо уже обрану вершину
6. Продовжуємо процес допоки не побудуємо каркас

Програмний код

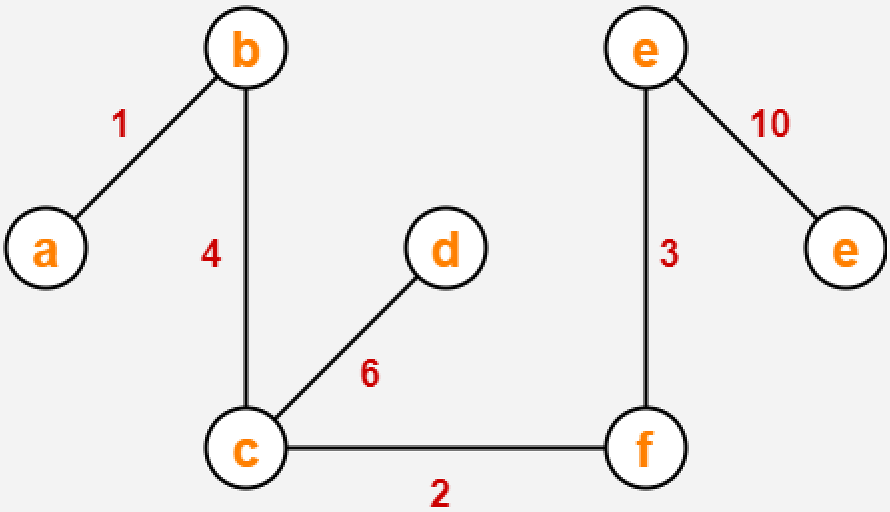
```python
def prima_algorithm(nodes, edges):
    """Implementation of Prima's algorithm."""
    nodes_len_initial = len(nodes)
    points = set([nodes[0]])
    carcas = set()
    total_points = 1

    while total_points < nodes_len_initial:
        smallest_path = (float('inf'), float('inf'), float('inf'))

        for edge in edges:
            if edge[2] <= smallest_path[2]:
                a = edge[0] in points
                b = edge[1] in points
                if (a and (not b)):
                    smallest_path = edge
                    e=1
                elif ((not a) and b):
                    smallest_path = edge
                    e=0

        carcas.add(smallest_path)
        points.add(smallest_path[e])
        edges.remove(smallest_path)
        total_points += 1

    return carcas
```

➢ Алгоритм Краскала

Даний алгоритм також є жадібним і його використовують для побудови мінімального кістякового дерева зваженого неорієнтованого графу.

Покроково:

1. Обираємо ребро e_1, яке має в графі найменшу вагу
2. Визначаємо послідовність ребер e_1 , e_2 , … , e_{n-1}
3. На кожному кроці обираємо ребро з найменшою вагою, яке не буде утворювати простих циклів з уже обраними ребрами
4. На вихід отримуємо мінімальний каркас початкового графа

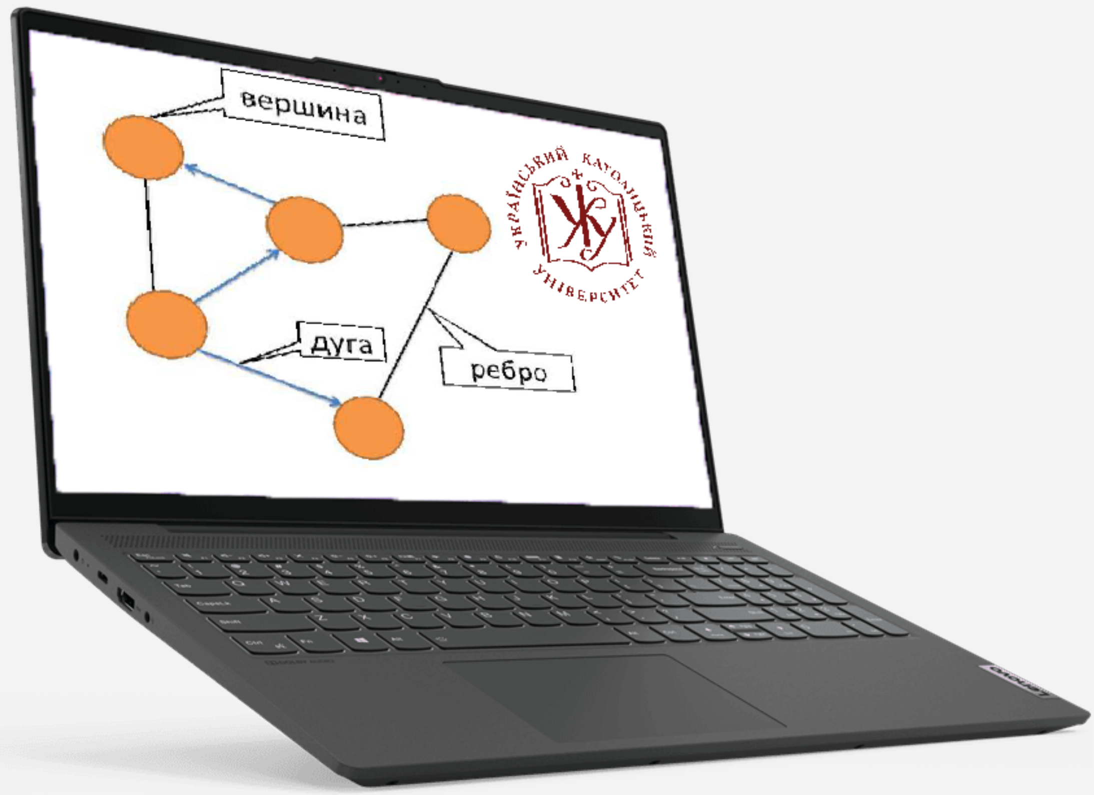


Програмний код

```python
def kruskal_algorithm(nodes, edges):
    """Implementation of Kruskal's algorithm."""
    edges = sorted(edges, key=lambda edge: edge[2])
    nodes_len_initial = len(nodes)
    points = set()
    carcas = set()
    total_edges = 0

    groups = []

    while total_edges < nodes_len_initial-1:
        edge = edges.pop(0)

        for group in groups:
            if (edge[0] in group) and (edge[1] in group):
                break

        else:
            if (edge[0] not in points) and (edge[1] not in points):

                groups.append([edge[0], edge[1]])
                points.add(edge[0])
                points.add(edge[1])

                carcas.add(edge)

                total_edges += 1

            elif ((edge[0] in points) and (edge[1] not in points)) or ((edge[0] not in points) and (edge[1] in points)):

                for group_n in range(len(groups)):
                    if (edge[0] in groups[group_n]) or (edge[1] in groups[group_n]):
                        groups[group_n] += [edge[0], edge[1]]

                        points.add(edge[0])
                        points.add(edge[1])
                        carcas.add(edge)

                        total_edges += 1
            else:
                group_a, group_a = None, None

                for group_n in range(len(groups)):
                    if edge[0] in groups[group_n]:
                        group_a = group_n

                for group_n in range(len(groups)):
                    if edge[1] in groups[group_n]:
                        group_b = group_n

                if (group_a is not None) and (group_a is not None):

                    indexes = (group_a, group_b)
                    for index in sorted(indexes, reverse=True):
                        if index == group_a:
                            group_aa = groups.pop(index)
                        if index == group_b:
                            group_bb = groups.pop(index)

                    groups.append(group_aa+group_bb)
                    carcas.add(edge)

                    total_edges += 1

    return carcas
```

Специфікація пристрою, на якому проводились експерименти

Кількість ядер	*4*
Тактова частота	*2.4 GHz*
Пам’ять	*16 gb RAM*
Операційна система	*Windows 10 Pro*



Програмний код експерименту

```
NUM_OF_ITERATIONS = 1000
NUM_OF_NODES = 100
```

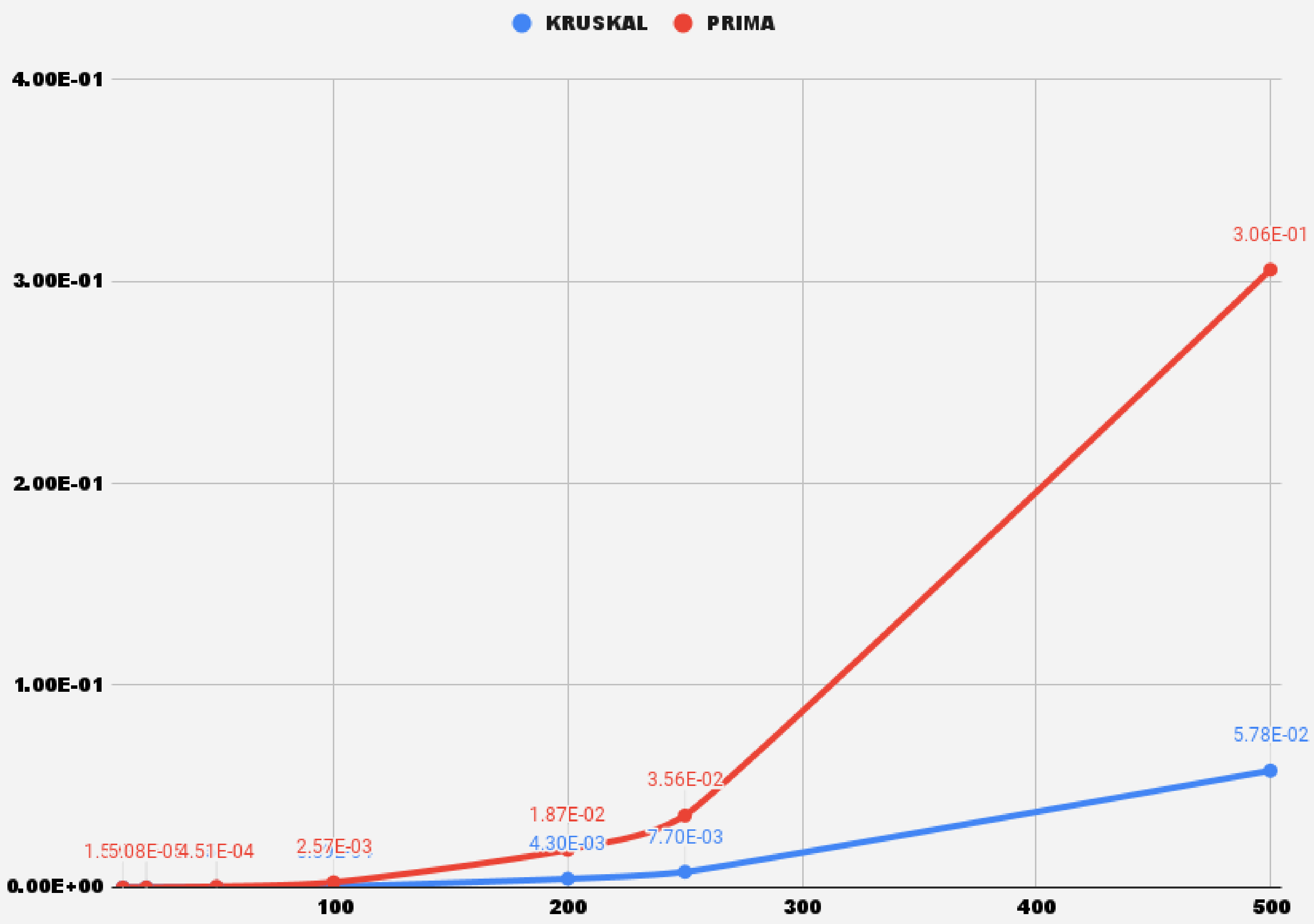
PRIM

```
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(NUM_OF_NODES, 0.1, False)

    # note that we should not measure time of graph convertion
    nodes = list(G.nodes)
    graph_edges = list(map(lambda x: (x[0], x[1], x[2]['weight']), G.edges().data()))

    start = time.time()
    carcas = prima_algorithm(nodes, graph_edges)
    end = time.time()

    # note that we should not measure time of graph creation
    G_prima_my = nx.Graph()
    G_prima_my.add_weighted_edges_from(carcas)

    time_taken += end - start

time_taken / NUM_OF_ITERATIONS
```

KRUSKAL

```
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(NUM_OF_NODES, 0.1, False)

    # note that we should not measure time of graph convertion
    nodes = list(G.nodes)
    graph_edges = list(map(lambda x: (x[0], x[1], x[2]['weight']), G.edges().data()))

    start = time.time()
    carcas = kruskal_algorithm(nodes, graph_edges)
    end = time.time()

    # note that we should not measure time of graph creation
    G_kruskal_my = nx.Graph()
    G_kruskal_my.add_weighted_edges_from(carcas)

    time_taken += end - start

time_taken / NUM_OF_ITERATIONS
```

Графік часу роботи алгоритмів



Результати експериментів

Після проведення експериментів для обчислення часу роботи алгоритмів Прима та Краскала із графами з різною кількістю вершин, результати яких можна спостерігати на графіку вище, ми зрозуміли, що реалізація алгоритму Краскала є більш успішною, дозволяючи йому виконувати обчислення швидше, ніж це робить алгоритм Прима. Жоден із алгоритмів не помиляється та не сповільнюється. Тестувались обидва на графах з різною кількістю вершин та було проведене в загальному кілька мільйонів ітерацій для аналізу їхньої швидкодії.

Висновок

У результаті виконання лабораторної роботи ми реалізували автоматичне обчислення мінімального каркасу зваженого графа використовуючи алгоритми Прима та Крускала. Провівши експерименти з 10, 20, 50, 100, 200, 250, 500 вершинами, ми дійшли висновку, що обидва алгоритми працюють дуже швидко, не помиляються та головне - не ламаються. Вони здатні обчислювати мінімальний каркас за дуже малий час, проте реалізація алгоритму Краскала в нашому випадку є більш вдалою, через що він здатний працювати значно швидше, аніж також реалізований нами алгоритм Прима.