Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

Выполнил студент группы КС-30 Положенцев Аксель Алексеевич

Ссылка на репозиторий: [MUCTR-IKT-CPP/PolozhencevAA\_30](https://github.com/MUCTR-IKT-CPP/PolozhencevAA_30)

Приняли: Крашенинников Роман Сергеевич

Дата сдачи: 17.02.2025

Москва 2025

Оглавление

[Описание задачи 3](#_Toc191294879)

[Описание метода 3](#_Toc191294880)

[Выполнение задачи 4](#_Toc191294881)

[Заключение 9](#_Toc191294882)

# Описание задачи

В данной работе требуется сгенерировать взвешенный неориентированный граф с заданным количеством вершин (10, 20, 50, 100). Каждая вершина должна быть соединена как минимум с определённым числом других вершин (3, 4, 10, 20), чтобы граф оставался связным. Вес рёбер задаётся случайным образом в диапазоне от 1 до 20. После генерации графа необходимо построить его минимальное остовное дерево с использованием алгоритма Краскала. Для каждого набора вершин проводится серия из нескольких тестов (5–10) с замером времени выполнения алгоритма, а затем строится график зависимости времени работы от количества вершин.

# Описание метода

**Для решения задачи используется алгоритм Краскала, который находит минимальное остовное дерево взвешенного неориентированного графа. Сначала создаётся граф, в котором вершины соединяются случайными рёбрами с учётом минимального количества связей, чтобы гарантировать связность. Затем алгоритм Краскала сортирует рёбра по весу и последовательно добавляет их в минимальное остовное дерево, избегая образования циклов. Для этого применяется структура "система непересекающихся множеств" (Union-Find), которая позволяет эффективно объединять компоненты связности. После выполнения алгоритма измеряется время его работы, а по итогам нескольких тестов усредняются результаты. Завершающим этапом является построение графика зависимости времени работы алгоритма от количества вершин.**

# Выполнение задачи

Был разработан код, включающий класс Graph для представления графа с возможностью добавления рёбер и вывода матрицы смежности. Реализована функция генерации случайного связного графа, а также алгоритм Краскала для построения минимального остовного дерева. В ходе экспериментов алгоритм тестировался на графах с различным количеством вершин (10, 20, 50, 100), причём для каждого размера проводилось по 5 тестов. В процессе тестирования результаты записывались в файл output.txt, включая матрицу смежности, найденное минимальное остовное дерево и время выполнения. По итоговым данным был построен график зависимости времени работы алгоритма от количества вершин.

**Код:**

import random

import time

import matplotlib.pyplot as plt

# для запуска python lab5.py

# Вариант 4. Построить минимальное остовное дерево взвешенного связного неориентированного графа с помощью алгоритма Краскала.

# ---------- Класс графа ----------

class Graph:

def \_\_init\_\_(self, num\_nodes: int):

"""Инициализация графа с заданным количеством узлов."""

self.num\_nodes = num\_nodes # Количество вершин в графе

self.edges = [] # Список рёбер в формате (узел\_1, узел\_2, вес)

self.adjacency\_matrix = [[0] \* num\_nodes for \_ in range(num\_nodes)] # Матрица смежности

def add\_edge(self, node1: int, node2: int, weight: int):

"""Добавление ребра в граф (в обе стороны, так как граф неориентированный)."""

self.edges.append((node1, node2, weight))

self.edges.append((node2, node1, weight)) # Дублируем для неориентированного графа

self.adjacency\_matrix[node1][node2] = weight

self.adjacency\_matrix[node2][node1] = weight

def generate\_random\_graph(self, min\_edges\_per\_node: int, weight\_range: tuple = (1, 20)):

"""Генерация случайного связного графа с минимумом рёбер на вершину."""

for i in range(self.num\_nodes - 1):

weight = random.randint(\*weight\_range)

self.add\_edge(i, i + 1, weight)

for node in range(self.num\_nodes):

num\_edges = random.randint(min\_edges\_per\_node, self.num\_nodes - 1)

connected\_nodes = set(v for u, v, \_ in self.edges if u == node) # Уже связанные узлы

while len(connected\_nodes) < num\_edges:

random\_node = random.randint(0, self.num\_nodes - 1)

if node != random\_node and random\_node not in connected\_nodes:

weight = random.randint(\*weight\_range)

self.add\_edge(node, random\_node, weight)

connected\_nodes.add(random\_node)

def print\_adjacency\_matrix(self):

"""Вывод матрицы смежности в консоль."""

for row in self.adjacency\_matrix:

print(" ".join(f"{weight:2}" for weight in row))

# Функция для поиска минимального остовного дерева (MST) с помощью алгоритма Краскала

def kruskal\_algorithm(graph: Graph):

"""Алгоритм Краскала для поиска минимального остовного дерева."""

sorted\_edges = sorted(graph.edges, key=lambda edge: edge[2]) # Сортируем рёбра по весу

parent = list(range(graph.num\_nodes)) # Родительские узлы для объединения

rank = [0] \* graph.num\_nodes # Ранги для балансировки объединения

mst\_result = [] # Список рёбер остовного дерева

def find\_root(node):

"""Функция поиска сжатия пути."""

if parent[node] != node:

parent[node] = find\_root(parent[node])

return parent[node]

def union\_sets(node1, node2):

"""Объединяет два множества."""

root1 = find\_root(node1)

root2 = find\_root(node2)

if root1 != root2:

if rank[root1] > rank[root2]:

parent[root2] = root1

elif rank[root1] < rank[root2]:

parent[root1] = root2

else:

parent[root2] = root1

rank[root1] += 1

# Обрабатываем рёбра и добавляем в MST, если они не образуют цикл

for node1, node2, weight in sorted\_edges:

if find\_root(node1) != find\_root(node2):

mst\_result.append((node1, node2, weight))

union\_sets(node1, node2)

return mst\_result

# Функция для тестирования алгоритма и записи результатов в файл

def test\_kruskal():

"""Функция для тестирования алгоритма Краскала и записи результатов."""

node\_sizes = [10, 20, 50, 100] # Различные размеры графов

min\_edges\_list = [3, 4, 10, 20] # Минимальное количество рёбер на вершину

test\_iterations = 5 # Количество запусков для усреднения времени

test\_results = [] # Хранение результатов

output\_file = "output.txt" # Имя файла для сохранения результатов

with open(output\_file, "w", encoding="utf-8") as file:

for num\_nodes, min\_edges in zip(node\_sizes, min\_edges\_list):

execution\_times = [] # Список для хранения времени выполнения

for \_ in range(test\_iterations):

# Генерируем случайный граф

test\_graph = Graph(num\_nodes)

test\_graph.generate\_random\_graph(min\_edges)

# Записываем матрицу смежности в файл

file.write(f"\nГраф с {num\_nodes} вершинами:\n")

for row in test\_graph.adjacency\_matrix:

file.write(" ".join(f"{weight:2}" for weight in row) + "\n")

# Запускаем алгоритм Краскала и замеряем время

start\_time = time.time()

mst = kruskal\_algorithm(test\_graph)

elapsed\_time = time.time() - start\_time

execution\_times.append(elapsed\_time)

# Записываем MST и время выполнения в файл

file.write(f"\nМинимальное остовное дерево (MST): {mst}\n")

file.write(f"Время выполнения: {elapsed\_time:.5f} сек\n")

# Усредняем время выполнения

avg\_time = sum(execution\_times) / len(execution\_times)

test\_results.append((num\_nodes, avg\_time))

# Записываем итоговые средние результаты в файл

file.write("\nСреднее время выполнения:\n")

for nodes, avg in test\_results:

file.write(f"{nodes} вершин: {avg:.5f} сек\n")

# Построение графика времени выполнения

plt.figure(figsize=(10, 6))

plt.plot([r[0] for r in test\_results], [r[1] for r in test\_results], label="Алгоритм Краскала", marker="o")

plt.xlabel("Количество вершин (N)")

plt.ylabel("Время выполнения (секунды)")

plt.title("Производительность алгоритма Краскала")

plt.legend()

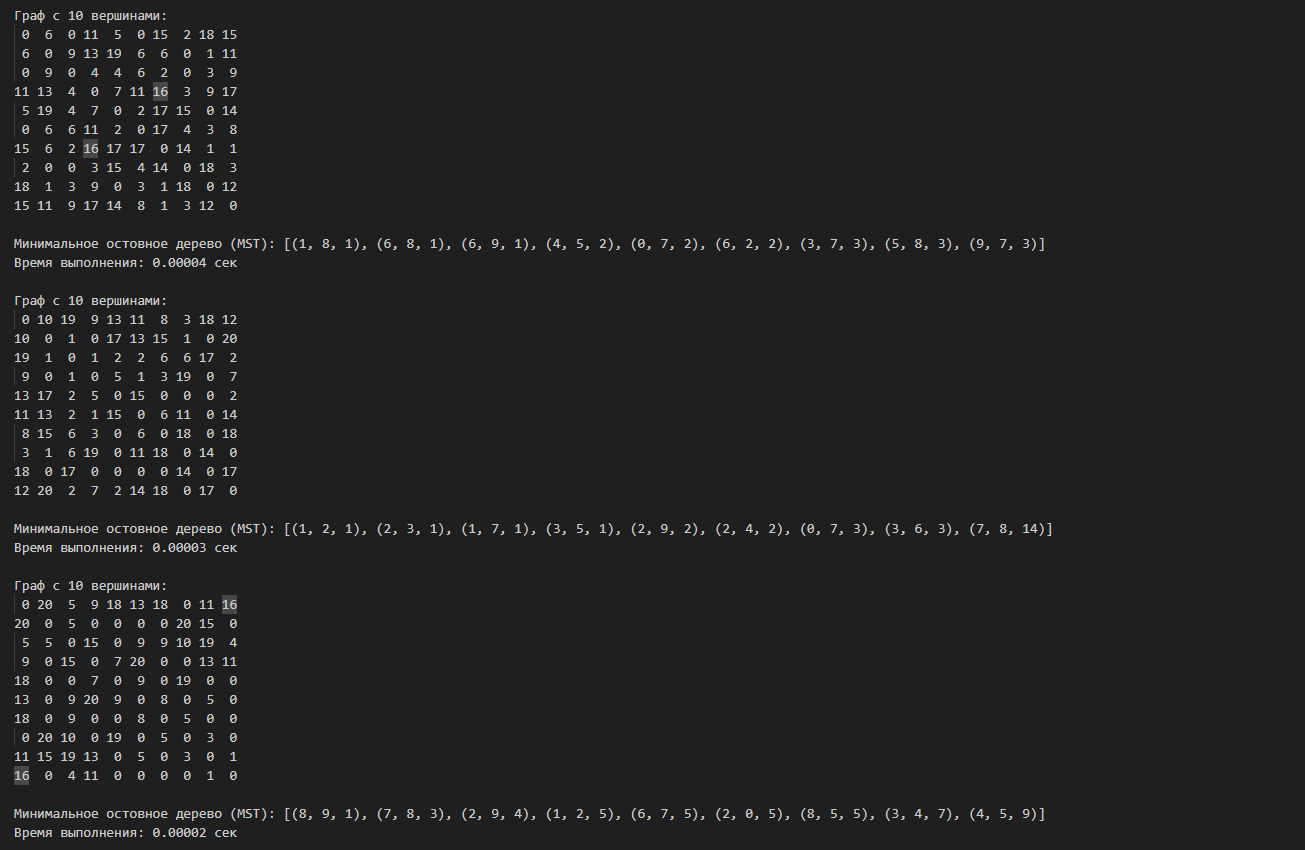
plt.grid()

plt.show()

# Запуск тестов

test\_kruskal()

**Результат выполнения программы:**



# Заключение

В результате проведённой работы был успешно реализован алгоритм Краскала, а также разработана система генерации случайных связных графов. Экспериментальные замеры показали, что время выполнения алгоритма увеличивается с ростом количества вершин, но остаётся в пределах предсказуемого порядка. Построенный график подтвердил эффективность алгоритма Краскала, демонстрируя его близкую к линейной зависимость времени работы от числа вершин. Таким образом, цели лабораторной работы были достигнуты, алгоритм успешно протестирован, а его производительность проанализирована

