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-36 Тернолуцкий Виктор Александрович

Приняли:

# Крашенинников Роман Сергеевич

Дата сдачи: 24.03.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

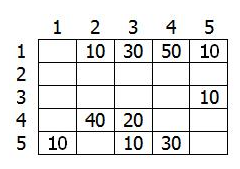
[Описание метода/модели.](#_Toc63548273) 3

[Выполнение задачи.](#_Toc63548274) 4

[Заключение. 2](#_Toc63548275)

# Описание задачи.

1. Создайте взвешенный граф, состоящий из [10, 20, 50, 100] вершин. Каждая вершина графа связана со случайным количеством вершин, минимум с [3, 4, 10, 20]. Веса ребер задаются случайным значением от 1 до 20. Каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, не обязательно прямой. (Можно использовать генератор из предыдущей лабораторной работы.)
2. Выведите получившийся граф в виде матрицы смежности. Пример вывода данных:

Матрица смежности

1. Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста., необходимо: Построить минимальное остовное дерево взвешенного связного неориентированного графа с помощью алгоритма Прима.
2. В рамках каждого теста, необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончанию всех тестов необходимо построить график используя полученные замеры времени, где на ось абсцисс (Х) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

# Описание метода/модели.

**Алгоритм Прима** для построение минимального островного дерева начинает обход с одной верщины и создает дерево, добавляя по одному ребру, до тех пор пока не будут включены все вершины.

Минимальное-Островное-Дерево-Прима (Граф)

Выбираем произвольную вершину, с которой начинается построение дерева

До тех пор пока остаются вершины не включенные в дерево

Выбираем ребро минимального веса между деревом и вершиной вне дерева

Добавляем выбранное ребро и вершину в итоговое дерево

Такой подход к решению задачи называется жадным алгоритмом, так как оно выбирает лучшее локальное решение игнорируя общую структуру дерева, что чревато издержками на длительность обхода этой структуры.

Жадные алгоритмы могут давать неправильные результаты, для доказательства правильности приведенного алгоритма используется доказательство от обратного, в котором считаем что на некотором шаге произошел неверный выбор, который мог произойти только при равенстве альтернатив, в противном случае был бы выбран другой вариант.

Асимптотическая сложность алгоритма O(n^2)

# Выполнение задачи.

Для реализации алгоритма использовался язык программирования Python.

**КОД**

mport random

import time

import heapq

import matplotlib.pyplot as plt

def generate\_graph(n, min\_edges):

matrix = [[0] \* n for \_ in range(n)]

# Создание связного графа (дерева)

added = {0}

remaining = set(range(1, n))

while remaining:

u = random.choice(list(added))

v = random.choice(list(remaining))

weight = random.randint(1, 20)

matrix[u][v] = weight

matrix[v][u] = weight

added.add(v)

remaining.remove(v)

# Добавление рёбер до достижения минимальной степени

for \_ in range(n \* 2): # Ограничение числа попыток

all\_ok = True

for u in range(n):

current\_degree = sum(1 for x in matrix[u] if x != 0)

if current\_degree < min\_edges:

all\_ok = False

possible = [v for v in range(n) if v != u and matrix[u][v] == 0]

needed = min\_edges - current\_degree

if len(possible) < needed:

needed = len(possible)

if needed == 0:

continue

selected = random.sample(possible, needed)

for v in selected:

weight = random.randint(1, 20)

matrix[u][v] = weight

matrix[v][u] = weight

if all\_ok:

break

return matrix

def prim\_optimized(matrix):

n = len(matrix)

key = [float('inf')] \* n

parent = [-1] \* n

key[0] = 0

heap = []

heapq.heappush(heap, (0, 0))

in\_tree = set()

while heap:

weight, u = heapq.heappop(heap)

if u in in\_tree:

continue

in\_tree.add(u)

for v in range(n):

if matrix[u][v] != 0 and v not in in\_tree and matrix[u][v] < key[v]:

key[v] = matrix[u][v]

parent[v] = u

heapq.heappush(heap, (key[v], v))

edges = []

for v in range(1, n):

if parent[v] != -1:

edges.append((parent[v], v, key[v]))

return edges

def print\_adjacency\_matrix(matrix):

print("Матрица смежности:")

for row in matrix:

print(' '.join(map(str, row)))

def main():

sizes = [10, 20, 50, 100]

min\_edges\_list = [3, 4, 10, 20]

tests\_per\_size = 5

time\_results = {n: [] for n in sizes}

for n, min\_edges in zip(sizes, min\_edges\_list):

print(f"\nОбработка графа с {n} вершинами...")

for test in range(tests\_per\_size):

print(f"Тест {test + 1}/{tests\_per\_size}")

# Генерация графа

start\_gen = time.time()

graph = generate\_graph(n, min\_edges)

end\_gen = time.time()

print(f"Время генерации: {end\_gen - start\_gen:.2f} сек")

if n <= 10 and test == 0: # Печать матрицы для малых графов

print\_adjacency\_matrix(graph)

# Замер времени алгоритма Прима

start\_prim = time.time()

prim\_optimized(graph)

end\_prim = time.time()

time\_taken = end\_prim - start\_prim

time\_results[n].append(time\_taken)

print(f"Время Прима: {time\_taken:.4f} сек")

avg\_time = sum(time\_results[n]) / tests\_per\_size

time\_results[n] = avg\_time

# Построение графика

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

plt.plot(sizes, [time\_results[n] for n in sizes], marker='o', linestyle='-', color='b')

plt.xlabel('Количество вершин (N)')

plt.ylabel('Среднее время выполнения (сек)')

plt.title('Зависимость времени выполнения алгоритма Прима от размера графа')

plt.grid(True)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Результаты расчётов:**

Обработка графа с 10 вершинами...

Тест 1/5

Время генерации: 0.00 сек

Матрица смежности:

0 0 17 10 0 0 0 0 11 12

0 0 11 0 15 4 3 0 0 0

17 11 0 0 16 0 0 0 0 0

10 0 0 0 0 0 16 8 12 0

0 15 16 0 0 20 0 1 10 15

0 4 0 0 20 0 10 0 0 0

0 3 0 16 0 10 0 0 0 0

0 0 0 8 1 0 0 0 15 4

11 0 0 12 10 0 0 15 0 0

12 0 0 0 15 0 0 4 0 0

Время Прима: 0.0000 сек

Тест 2/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 3/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 4/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 5/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Обработка графа с 20 вершинами...

Тест 1/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 2/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 3/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 4/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Тест 5/5

Время генерации: 0.00 сек

Время Прима: 0.0000 сек

Обработка графа с 50 вершинами...

Тест 1/5

Время генерации: 0.00 сек

Время Прима: 0.0001 сек

Тест 2/5

Время генерации: 0.00 сек

Время Прима: 0.0001 сек

Тест 3/5

Время генерации: 0.00 сек

Время Прима: 0.0002 сек

Тест 4/5

Время генерации: 0.00 сек

Время Прима: 0.0001 сек

Тест 5/5

Время генерации: 0.00 сек

Время Прима: 0.0001 сек

Обработка графа с 100 вершинами...

Тест 1/5

Время генерации: 0.00 сек

Время Прима: 0.0005 сек

Тест 2/5

Время генерации: 0.00 сек

Время Прима: 0.0005 сек

Тест 3/5

Время генерации: 0.00 сек

Время Прима: 0.0005 сек

Тест 4/5

Время генерации: 0.00 сек

Время Прима: 0.0005 сек

Тест 5/5

Время генерации: 0.00 сек

Время Прима: 0.0004 сек

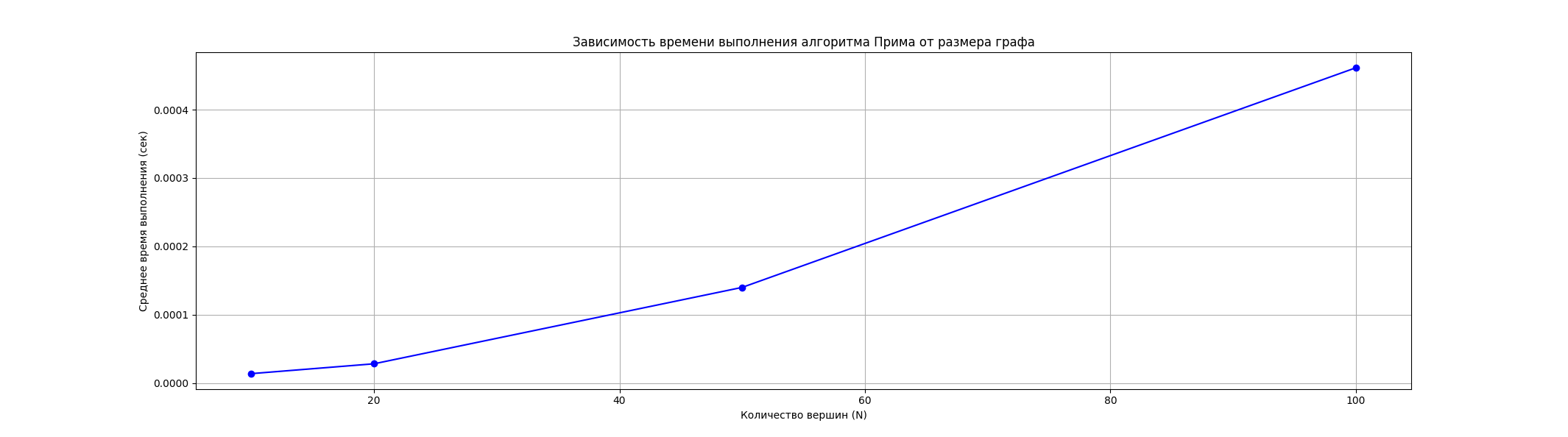


Рисунок 1 – Зависимость времени от количества вершин графа

**Заключение**

Результаты проведенных тестов демонстрируют, что алгоритм Прима эффективно справляется с построением минимального остовного дерева для графов малого и среднего размера (от 10 до 100 вершин). Анализ времени выполнения показывает, что с увеличением числа вершин время работы алгоритма растет, что соответствует его теоретической асимптотической сложности **O(n²).** На графике видно, что при переходе от 10 к 100 вершинам время увеличивается нелинейно, но остается в разумных пределах, что подтверждает практическую применимость алгоритма для графов умеренного размера.

Увеличение вычислительных затрат объясняется тем, что на каждом шаге алгоритм должен проверять все рёбра, соединяющие уже включённые в дерево вершины с оставшимися. Это приводит к квадратичной зависимости времени от количества вершин. Таким образом, алгоритм Прима хорошо подходит для решения задач на графах с числом вершин до нескольких сотен, но для более крупных графов может потребоваться оптимизация (например, использование приоритетной очереди с более эффективной структурой данных).