МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Ка­фед­ра Диф­фе­рен­ци­аль­ных урав­не­ний, ма­те­ма­ти­че­ско­го и чис­лен­но­го ана­ли­за**

**Отчет по учебной практике**

**«Нахождение кратчайших путей в графе»**

**Выполнил:** студент группы 381906-1

Яшин Кирилл Евгеньевич

\_\_\_\_\_\_\_\_\_\_\_

**Принял:** Малышев Дмитрий Сергеевич

\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

2021.

Содержание

[1. Введение 3](#_Toc84983973)

[2. Постановка задачи 4](#_Toc84983974)

[3. Руководство пользователя 5](#_Toc84983975)

[4. Руководство программиста 6](#_Toc84983976)

[5. Исследование эффективности 8](#_Toc84983977)

[6. Заключение 12](#_Toc84983978)

# Введение

**Основная цель данной работы** – сравнение эффективности двух алгоритмов для поиска кратчайшего пути в графе – алгоритма Дейкстры на основе меток и алгоритма Форда-Беллмана.

**Алгоритм Дейкстры** – алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

Суть алгоритма довольно проста. Каждой вершине из **V** сопоставим метку — минимальное известное расстояние от этой вершины до **a**. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Метка самой вершины **a** полагается равной 0, метки остальных вершин — бесконечности. Все вершины графа помечаются как непосещённые. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина **u**, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из **u**, назовём соседями этой вершины. Для каждого соседа вершины **u**, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки **u** и длины ребра, соединяющего **u** с этим соседом.

Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину **u** как посещённую и повторим [шаг алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B#%D0%A8%D0%B0%D0%B3).

**Алгоритм Форда-Беллмана** – алгоритм нахождения кратчайшего пути от одной вершины графа до всех остальных. Мы считаем, что граф не содержит цикла отрицательного веса. Случай наличия отрицательного цикла будет рассмотрен ниже в отдельном разделе.

Заведём массив расстояний **d[0, n – 1]** который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: **d[a] = 0**, а все остальные элементы равны бесконечности.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию вдоль каждого ребра **(a, b)**стоимости **c**. Релаксация вдоль ребра — это попытка улучшить значение **d[b]**значением **d[a] + c**.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Реализация генератора графов.
2. Реализация алгоритма Дейкстры, использующего метки.
3. Реализация алгоритма Форда-Беллмана.
4. Сравнение эффективности двух алгоритмов в зависимости от параметров графа.

# Руководство пользователя

В проекте есть три python-скрипта:

1. algs – вспомогательный модуль с функциями генерации графа и алгоритмами
2. time\_test – скрипт, осуществляющий расчеты и считающий время работы алгоритмов
3. plots – скрипт, который рисует графики зависимостей времени выполнения от параметров графа

# Руководство программиста

В данном разделе приводится код алгоритмов.

def generator(possibility: float, size, q, r: int) -> set:  
 \_edges = set()  
 vertexes = set([v for v in range(size)])  
  
 for combination in combinations(vertexes, 2):  
 temp = random.random()  
 weight = random.randint(q, r)  
 if temp < possibility:  
 current\_edge = (combination[0], combination[1], weight)  
 \_edges.add(current\_edge)  
  
 return \_edges

*Генерация графа*

def dijkstra(\_adj\_matrix: list, start, size: int) -> [list, list]:  
 int\_max = 10\*\*10  
 distance = [int\_max] \* size  
 visited = [False] \* size  
 distance[start] = 0  
 last\_visited = [-1] \* size  
  
 for \_ in range(size - 1):  
 minimum, index = int\_max, -1  
 for current in range(size):  
 if not visited[current] and distance[current] <= minimum:  
 minimum, index = distance[current], current  
 visited[index] = True  
  
 for i in range(size):  
 if not visited[i] and \_adj\_matrix[index][i] and distance[index] != int\_max \  
 and (distance[index] + \_adj\_matrix[index][i] < distance[i]):  
 distance[i] = distance[index] + \_adj\_matrix[index][i]  
 last\_visited[i] = index  
  
 for i in range(len(distance)):  
 if distance[i] == int\_max:  
 distance[i] = None  
 last\_visited[i] = None  
  
 return distance, last\_visited

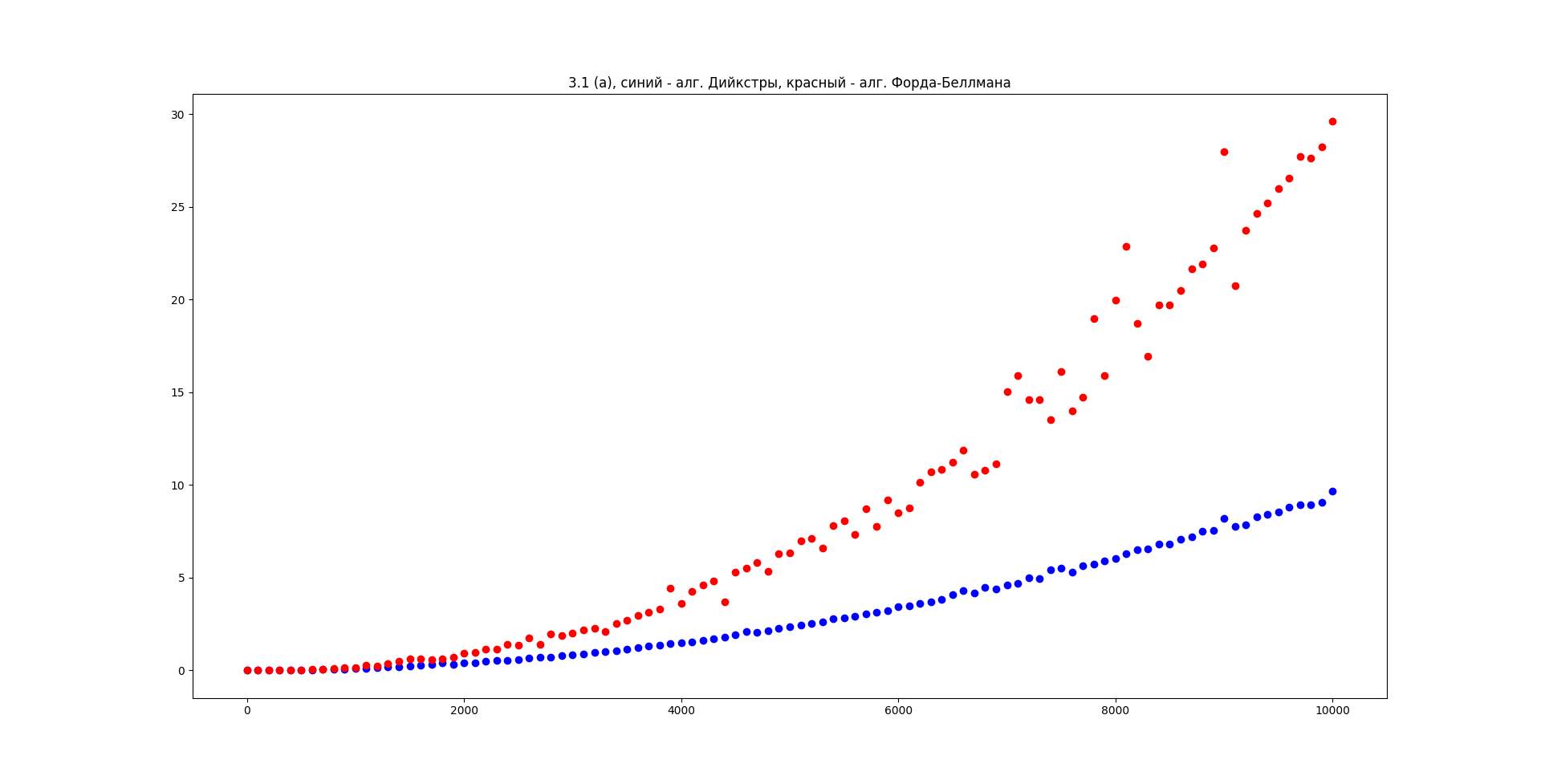
*Алгоритм Дейкстры*

def ford\_bellman(\_edges: set, start, size: int) -> [list, list]:  
 int\_max = 10\*\*10  
 distance = [int\_max] \* size  
 distance[start] = 0  
 last\_visited = [-1] \* size  
 cnt = 1  
 stop = False  
 while cnt < size and not stop:  
 cnt += 1  
 stop = True  
 for \_edge in \_edges:  
 if distance[\_edge[0]] + \_edge[2] < distance[\_edge[1]]:  
 distance[\_edge[1]] = distance[\_edge[0]] + \_edge[2]  
 stop = False  
 last\_visited[\_edge[1]] = \_edge[0]  
  
 for i in range(len(distance)):  
 if distance[i] == int\_max:  
 distance[i] = None  
 last\_visited[i] = None  
  
 return distance, last\_visited

*Алгоритм Форда-Беллмана*

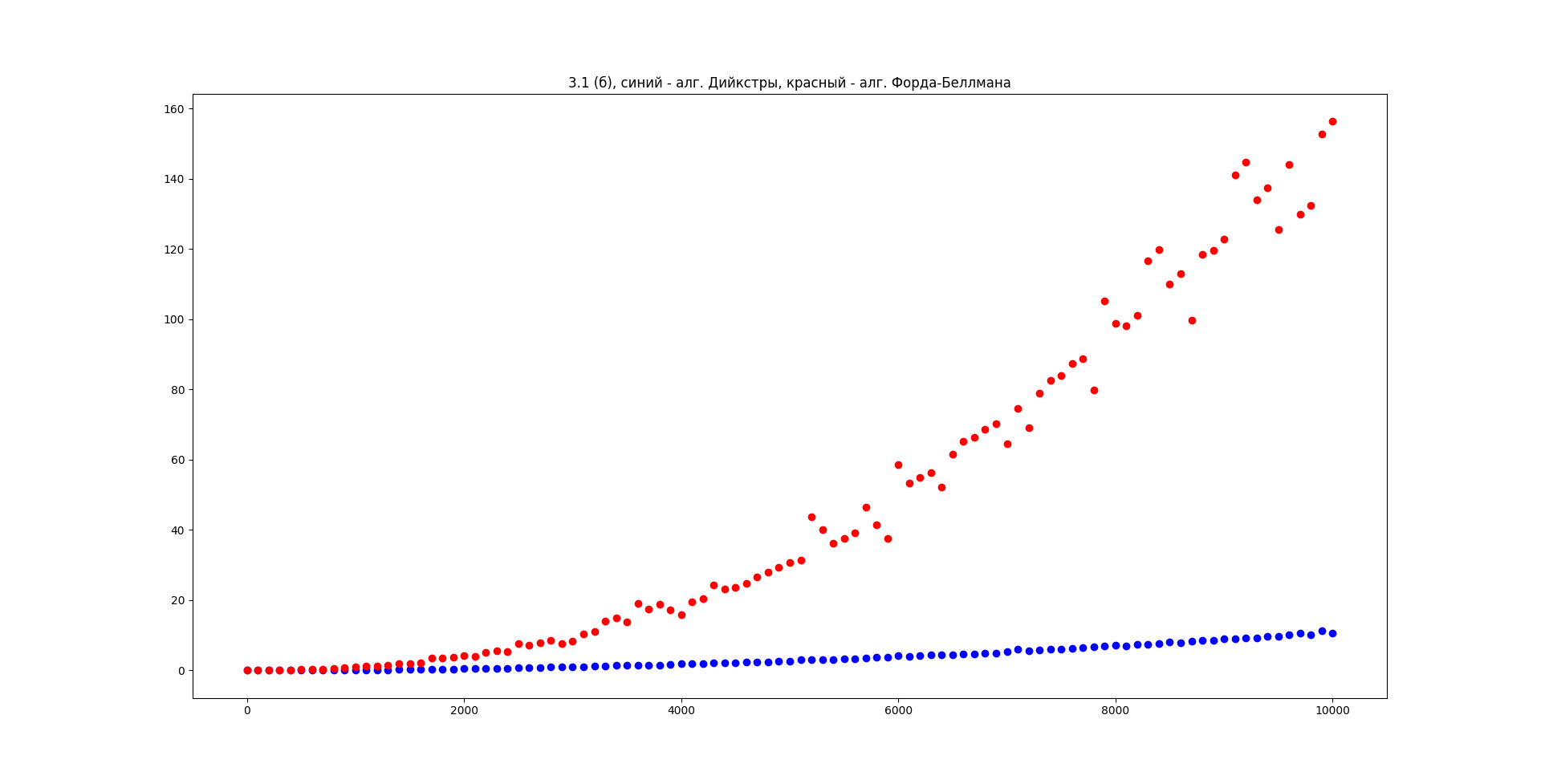
# Исследование эффективности

* + 1. **Число вершин от 1 до 104 + 1 с шагом 100, число ребер ≈ n2 / 10**



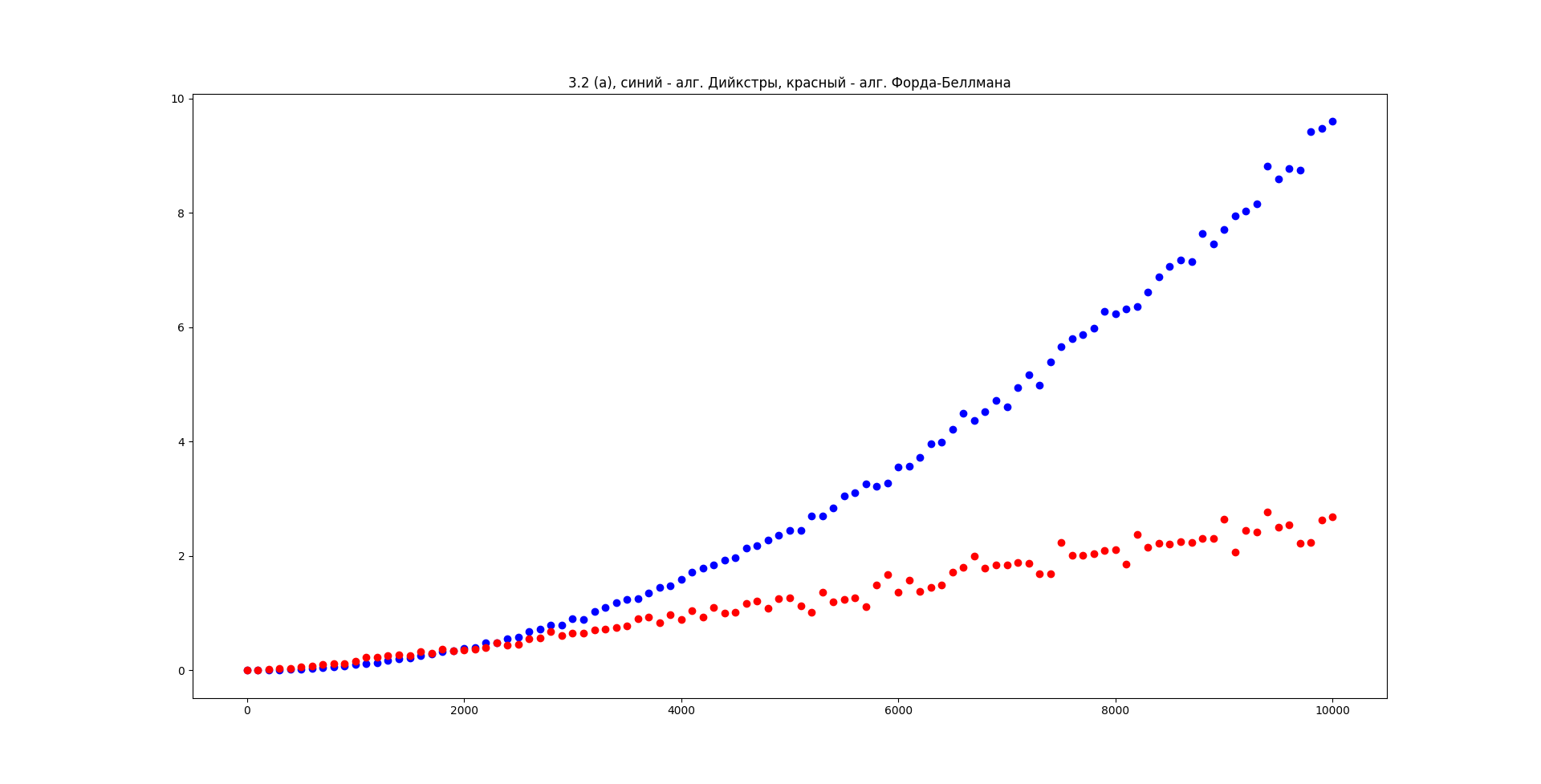
Легко сделать вывод о том, что при увеличении числа вершин алгоритм Форда-Беллмана проигрывает в эффективности. Это связано с тем, что его сложность – O(mn), то есть при m = n2 / 10, O(n3). У алгоритма Дейкстры же сложность квадратичная.

* + 1. **Число вершин от 1 до 104 + 1 с шагом 100, число ребер ≈ n2**



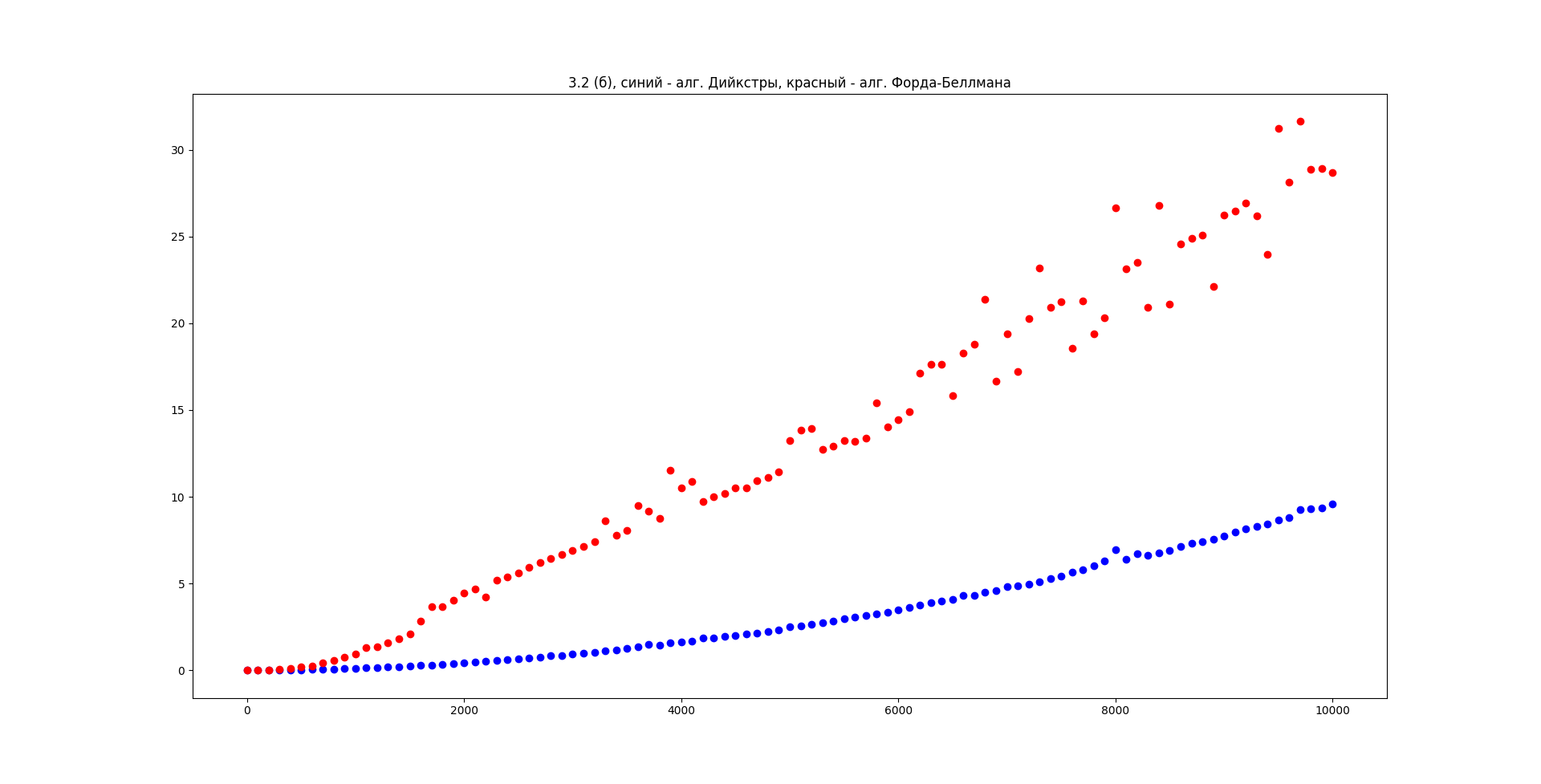
В данном примере наблюдаются те же зависимости, только время выполнения алгоритма Форда-Беллмана еще хуже. Это связано с тем, что ребер в графе примерно в 10 раз больше.

* + 1. **Число вершин от 1 до 104 + 1 с шагом 100, число ребер ≈ 100n**



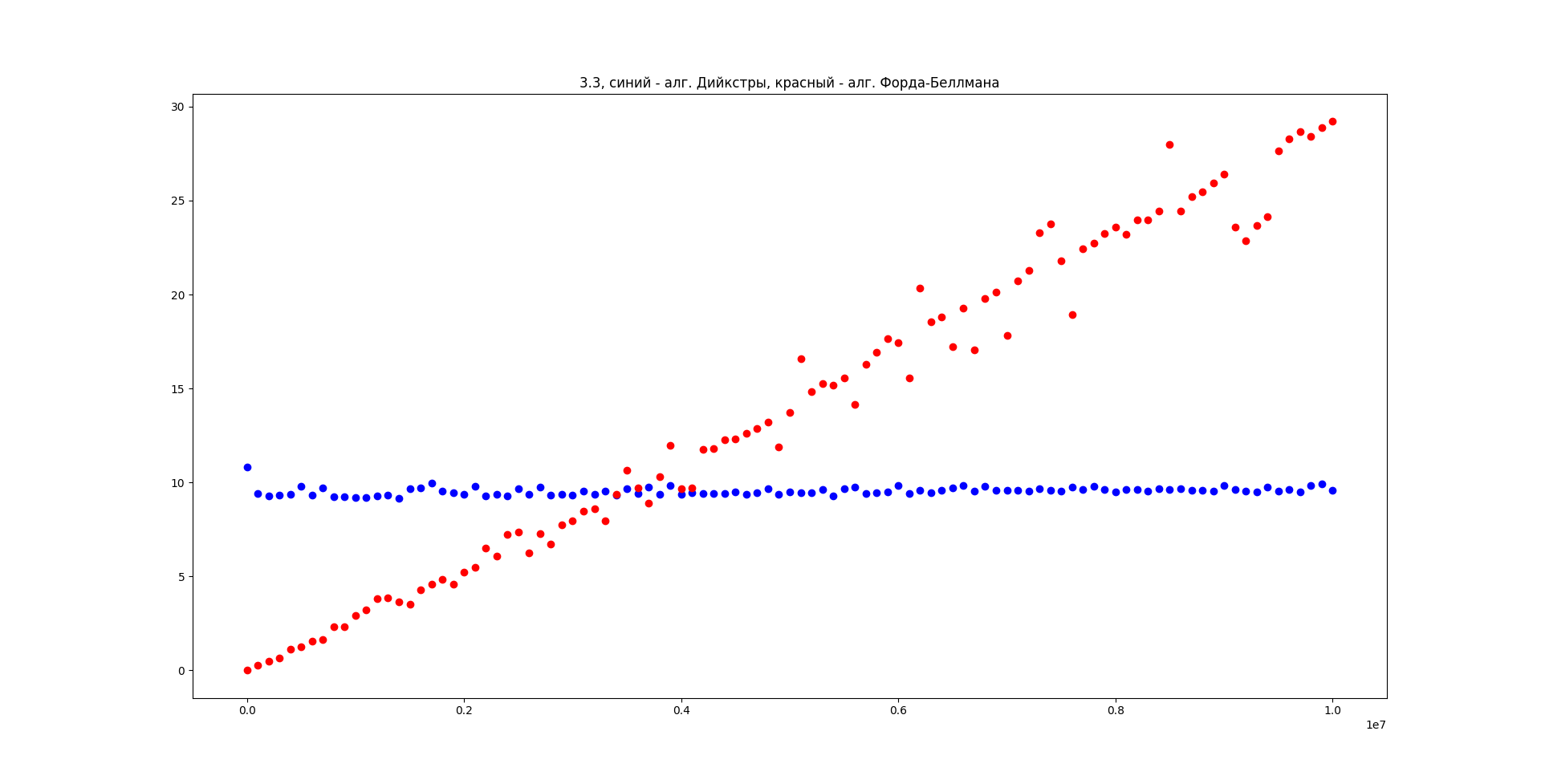
В данном случае хорошо видно, что при малых значениях числа ребер алгоритм Форда-Беллмана оказывается эффективнее.

* + 1. **Число вершин от 1 до 104 + 1 с шагом 100, число ребер ≈ 1000n**



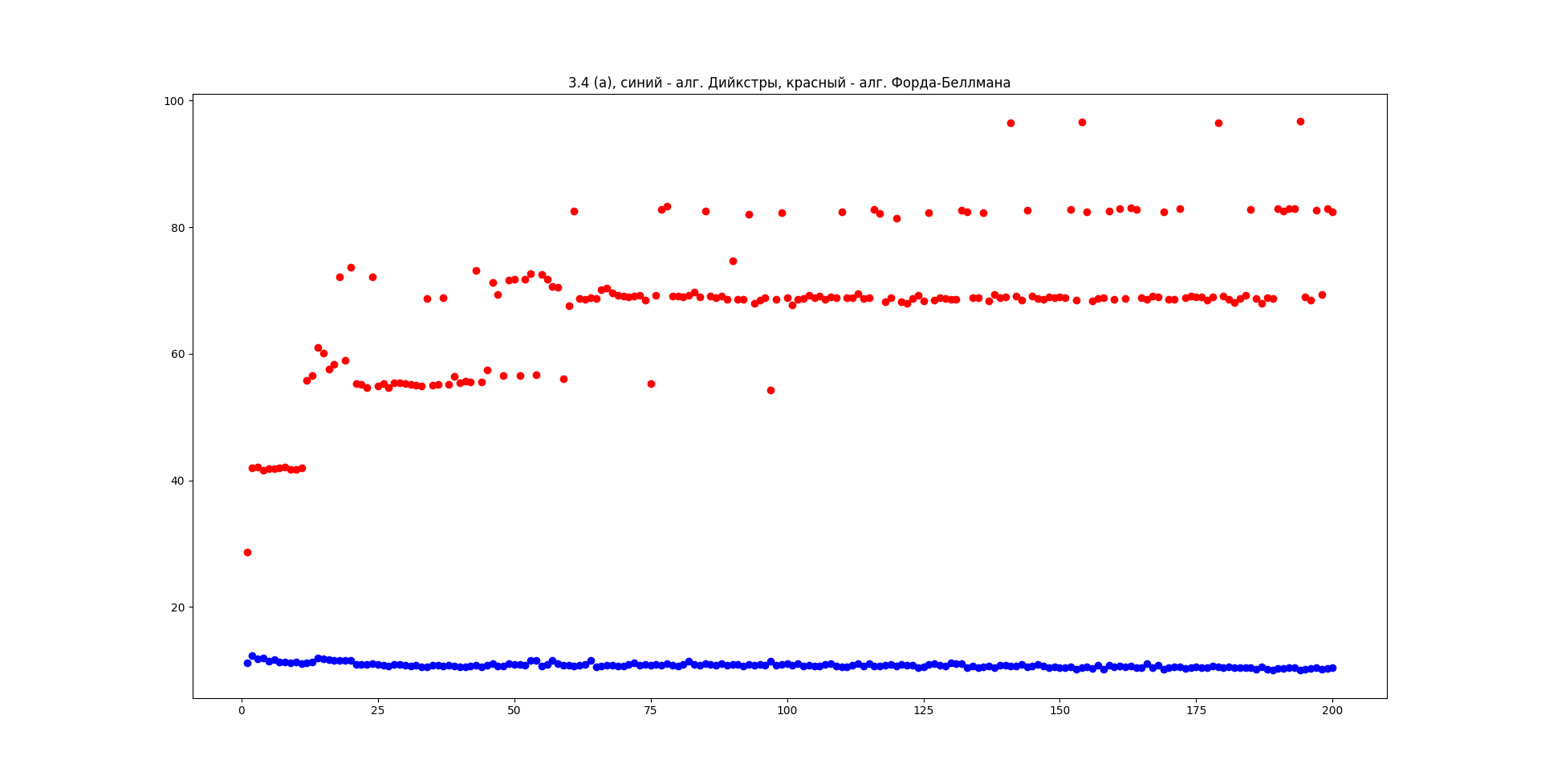
Ребер стало больше, что оказало влияние на скорость работы алгоритма Форда-Беллмана, алгоритм Дейкстры же стабилен.

* + 1. **Число вершин - 104 + 1, число ребер меняется от 0 до 107 с шагом 105**



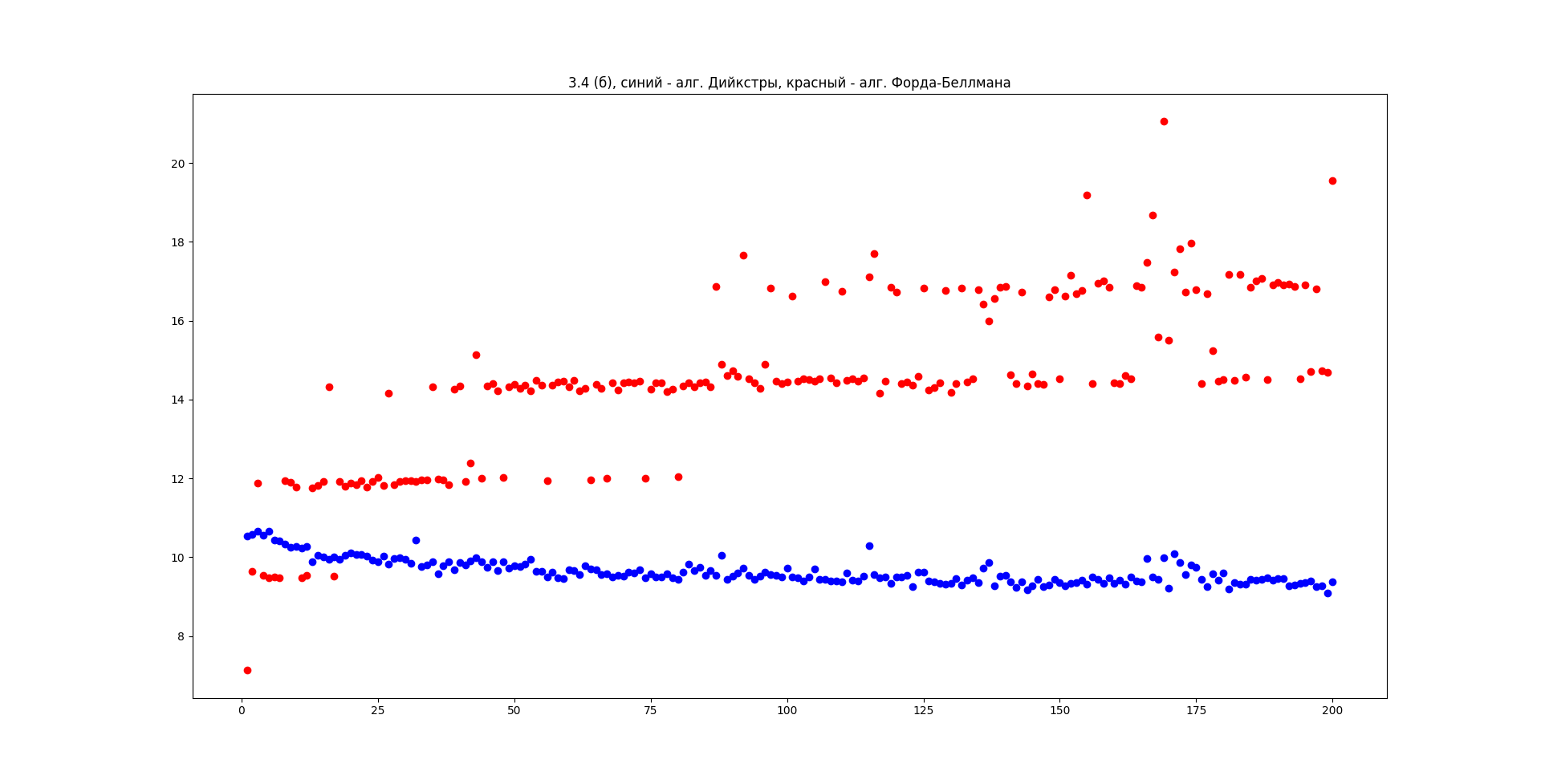
Алгоритм Дейкстры не зависит от числа ребер, поэтому время его выполнения постоянно. По графику для алгоритма Форда-Беллмана можно с легкостью проследить линейную зависимость между числом ребер и временем выполнения алгоритма.

* + 1. **Число 1 до 104 + 1, число ребер ≈ n2, диапазон весов от 1 до r, r изменяется от 1 до 200 с шагом 1**



Видно, что изменение весов ребер влияет только на алгоритм Форда-Беллмана. Это связано с тем, что в его реализации больше сравнений и математических операций.

* + 1. **Число 1 до 104 + 1, число ребер ≈ n2** / **10, диапазон весов от 1 до r, r изменяется от 1 до 200 с шагом 1**



Стало меньше ребер, время работы алгоритма Форда-Беллмана уменьшилось, что логично следует из оценки сложности алгоритма.

# Заключение

Эта лабораторная работа дала возможность сравнить эффективность двух алгоритмов поиска кратчайшего пути от вершины графа до остальных вершин.

Анализируя полученные результаты, можно сделать вывод о том, что алгоритм Дейкстры более надежен и предсказуем с точки зрения времени работы. В большинстве случаев он оказался эффективнее алгоритма Форда-Беллмана. Я рекомендую использовать алгоритм Форда-Беллмана только в случае малого числа ребер в графе, только тогда он оказывается эффективным.

1. **Литература**
2. <https://e-maxx.ru/algo/ford_bellman>
3. <https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры>
4. Алгоритмы. Построение и анализ. – Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн