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 № 4

Выполнил студент группы КС-36 Кошкарев Иван Михайлович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/KoshkarevIM\_36\_ALG/tree/main/Lab1

Приняли: Крашенинников Роман Сергеевич

Дата сдачи: 17.03

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

Реализовать модель грфаф, а также добавить к ней методы выводов различном виде. Реализовать функции генерации графов, поиска в ширину, поиска в глубину. Сравнить разные методы поиска

# Описание метода/модели.

Под графом в математике понимается абстракция реальной системы объектов безотносительно их природы, обладающих парными связями.

Вершина графа – это некоторая точка связанная с другими точками

Ребро графа – это линия, соединяющая две точки и олицетворяющая связь между ними

Граф – это множество вершин соединённых друг с другом произвольным образом множеством ребер

Существуют ориентированные, сложные и простые графы,

Специальные виды графов:

Полный граф – все вершины графа соединены друг с другому

Регулярный граф – степени всех вершин одинаковы

Биграф – все вершины разделяются на две группы и нет такого ребра который соединяет вершины из одной группы

Дерево – это граф, в котором все вершины соединены лишь одним маршрутом

Графы можно описать несколькими способами:

Список ребер

Список смежности

Матрица смежности (столбцы и строки соответствуют вершинам)  
Матрица инциндетности (строки соответствуют вершинам, а столбцы соответствуют связям)

Способы обхода графа:

**DFS** (Deep first search) следует концепции «погружайся глубже, головой вперед» («go deep, head first»). Идея заключается в том, что мы двигаемся от начальной вершины (точки, места) в определенном направлении (по определенному пути) до тех пор, пока не достигнем конца пути или пункта назначения (искомой вершины). Если мы достигли конца пути, но он не является пунктом назначения, то мы возвращаемся назад (к точке разветвления или расхождения путей) и идем по другому маршруту.

**BFS** (Breadth first search) следует концепции «расширяйся, поднимаясь на высоту птичьего полета» («go wide, bird’s eye-view»). Вместо того, чтобы двигаться по определенному пути до конца, BFS предполагает движение вперед по одному соседу за разВместо следования по пути, BFS подразумевает посещение ближайших к s соседей за одно действие (шаг), затем посещение соседей соседей и так до тех пор, пока не будет обнаружено t

# Выполнение задачи.

Для реализации модели и тестов я использовал язык Python из-за его простой архитектуры.

РЕАЛИЗАЦИЯ КЛАССА И МЕТОДОВ ВЫВОДА

class Graph:  
 def \_\_init\_\_(self, num\_vert, edges, directed=False):  
 self.num\_vert = num\_vert  
 self.edges = edges  
 self.directed = directed  
  
 # Матрица смежности  
 def adjacency\_matrix(self):  
 matrix = [[0] \* self.num\_vert for \_ in range(self.num\_vert)]  
 for u, v in self.edges:  
 matrix[u][v] = 1  
 if not self.directed:  
 matrix[v][u] = 1  
 return matrix  
  
 # Матрица инцидентности  
 def incidence\_matrix(self):  
 matrix = [[0] \* len(self.edges) for \_ in range(self.num\_vert)]  
 for i, (u, v) in enumerate(self.edges):  
 matrix[u][i] = 1  
 if self.directed:  
 matrix[v][i] = -1  
 else:  
 matrix[v][i] = 1  
 return matrix  
  
 # Список смежности  
 def adjacency\_list(self):  
 adj\_list = {v: [] for v in range(self.num\_vert)}  
 for u, v in self.edges:  
 adj\_list[u].append(v)  
 if not self.directed:  
 adj\_list[v].append(u)  
 return adj\_list  
  
 # Список ребер  
 def edge\_list(self):  
 return self.edges

РЕАЛИЗАЦИЯ МЕТОДОВ ОБХОДА

# Поиск в ширину (BFS)  
def bfs(graph, start, end):  
 visited = set()  
 queue = deque([(start, [start])])  
  
 while queue:  
 node, path = queue.popleft()  
 if node == end:  
 return path  
 if node not in visited:  
 visited.add(node)  
 for neighbor in graph.adjacency\_list()[node]:  
 if neighbor not in visited:  
 queue.append((neighbor, path + [neighbor]))  
 return None  
  
  
# Поиск в глубину (DFS)  
def dfs(graph, start, end, visited=None, path=None):  
 if visited is None:  
 visited = set()  
 if path is None:  
 path = []  
  
 visited.add(start)  
 path = path + [start]  
  
 if start == end:  
 return path  
  
 for neighbor in graph.adjacency\_list()[start]:  
 if neighbor not in visited:  
 new\_path = dfs(graph, neighbor, end, visited, path)  
 if new\_path:  
 return new\_path  
 return None

РЕАЛИЗАЦИЯ ГЕНЕРАЦИИ СЛУЧАЙНЫХ ГРАФОВ

def generate\_random\_graph(min\_vert, max\_vert, min\_edges, max\_edges, max\_degree, directed=False,  
 max\_in\_degree=None, max\_out\_degree=None):  
 num\_vert = random.randint(min\_vert, max\_vert)  
 num\_edges = random.randint(min\_edges, max\_edges)  
  
 edges = []  
 degree = [0] \* num\_vert  
  
 for i in range(num\_edges):  
 u = random.randint(0, num\_vert - 1)  
 v = random.randint(0, num\_vert - 1)  
  
 if u == v:  
 i = i - 1  
 continue  
  
 if directed:  
 if max\_out\_degree and degree[u] >= max\_out\_degree:  
 i = i - 1  
 continue  
 if max\_in\_degree and degree[v] >= max\_in\_degree:  
 i = i - 1  
 continue  
 else:  
 if degree[u] >= max\_degree or degree[v] >= max\_degree:  
 i = i - 1  
 continue  
  
 edges.append((u, v))  
 degree[u] += 1  
 degree[v] += 1  
  
 return Graph(num\_vert, edges, directed)

РЕАЛИЗАЦИЯ ТЕСТОВ

# Параметры для генерации графов  
min\_ver = 10  
max\_ver = 100  
min\_edge = 20  
max\_edge = 200  
max\_degree = 5  
directed = False  
  
# Демонстрация работы методов класса  
  
# Генерация графа  
graph = generate\_random\_graph(min\_ver, max\_ver, min\_edge, max\_edge, max\_degree, directed)  
  
# Вывод информации о графе  
print("Матрица смежности:")  
for row in graph.adjacency\_matrix():  
 print(row)  
  
print("\nМатрица инцидентности:")  
for row in graph.incidence\_matrix():  
 print(row)  
  
print("\nСписок смежности:")  
for key, value in graph.adjacency\_list().items():  
 print(f"{key}: {value}")  
  
print("\nСписок ребер:")  
print(graph.edge\_list())  
  
# Списки для хранения результатов  
bfs\_times = []  
dfs\_times = []  
sizes = []  
  
# Генерация 10 графов с возрастающим количеством вершин и ребер  
for i in range(10):  
 num\_ver = 100 + i \* 200  
 num\_edge = 100 + i \* 400  
 graph = generate\_random\_graph(num\_ver, num\_ver, num\_edge, num\_edge, max\_degree, directed)  
 sizes.append(num\_ver)  
  
 # Выбор случайных вершин для поиска пути  
 start = random.randint(0, num\_ver - 1)  
 end = random.randint(0, num\_ver - 1)  
  
 # Замер времени выполнения BFS  
 start\_time = time.time()  
 bfs\_path = bfs(graph, start, end)  
 bfs\_time = time.time() - start\_time  
 bfs\_times.append(bfs\_time)  
  
 # Замер времени выполнения DFS  
 start\_time = time.time()  
 dfs\_path = dfs(graph, start, end)  
 dfs\_time = time.time() - start\_time  
 dfs\_times.append(dfs\_time)  
  
 # Вывод информации о графе и путях  
 print(f"Граф {i + 1}:")  
 print(f"Количество вершин: {num\_ver}, Количество ребер: {num\_edge}")  
 print("-" \* 50)  
  
# Построение графика  
plt.plot(sizes, bfs\_times, label='BFS', marker='o')  
plt.plot(sizes, dfs\_times, label='DFS', marker='o')  
plt.xlabel('Количество вершин')  
plt.ylabel('Время (секунды)')  
plt.title('Сравнение времени выполнения BFS и DFS')  
plt.legend()  
plt.grid(True)  
plt.show()

***Результаты:***

Матрица смежности:

[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0]

[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0]

[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]

[0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]

[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]

[0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]

[0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

Матрица инцидентности:

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1]

[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]

[0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Список смежности:

0: [3, 17, 6, 1, 16]

1: [10, 17, 0, 18]

2: [5, 10, 11, 13, 8]

3: [0, 5, 12, 12, 18]

4: [7, 19, 7, 17, 5]

5: [6, 3, 2, 11, 4]

6: [5, 12, 8, 11, 0]

7: [18, 4, 16, 9, 4]

8: [6, 2]

9: [7, 10, 14]

10: [1, 2, 9]

11: [14, 5, 2, 6]

12: [6, 3, 14, 3, 13]

13: [14, 18, 12, 2]

14: [13, 11, 12, 15, 9]

15: [14]

16: [7, 17, 0]

17: [19, 4, 1, 16, 0]

18: [7, 13, 3, 1]

19: [4, 17]

Список ребер:

[(5, 6), (18, 7), (12, 6), (7, 4), (0, 3), (1, 10), (14, 13), (16, 7), (4, 19), (5, 3), (8, 6), (13, 18), (9, 7), (7, 4), (17, 19), (12, 3), (5, 2), (17, 4), (10, 2), (11, 14), (12, 14), (12, 3), (13, 12), (11, 5), (4, 5), (10, 9), (3, 18), (2, 11), (17, 1), (14, 15), (17, 16), (17, 0), (14, 9), (11, 6), (0, 6), (1, 0), (2, 13), (2, 8), (18, 1), (16, 0)]

Граф 1:

Количество вершин: 100, Количество ребер: 100

--------------------------------------------------

Граф 2:

Количество вершин: 300, Количество ребер: 500

--------------------------------------------------

Граф 3:

Количество вершин: 500, Количество ребер: 900

--------------------------------------------------

Граф 4:

Количество вершин: 700, Количество ребер: 1300

--------------------------------------------------

Граф 5:

Количество вершин: 900, Количество ребер: 1700

--------------------------------------------------

Граф 6:

Количество вершин: 1100, Количество ребер: 2100

--------------------------------------------------

Граф 7:

Количество вершин: 1300, Количество ребер: 2500

--------------------------------------------------

Граф 8:

Количество вершин: 1500, Количество ребер: 2900

--------------------------------------------------

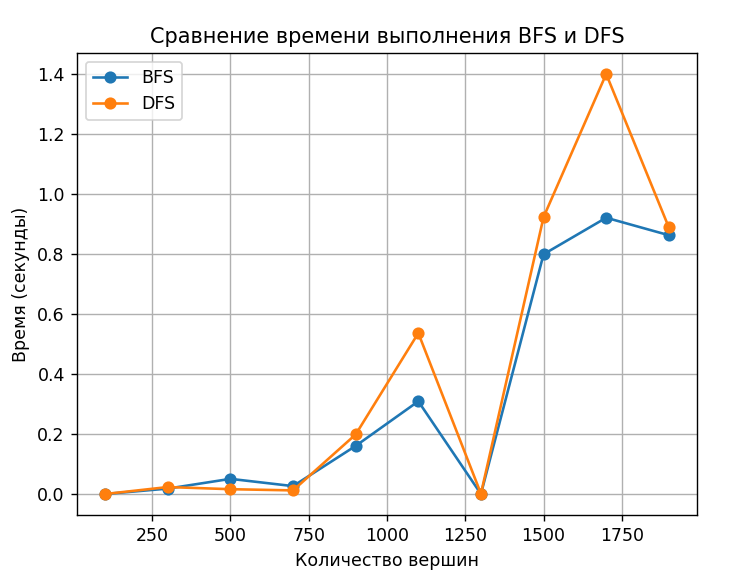
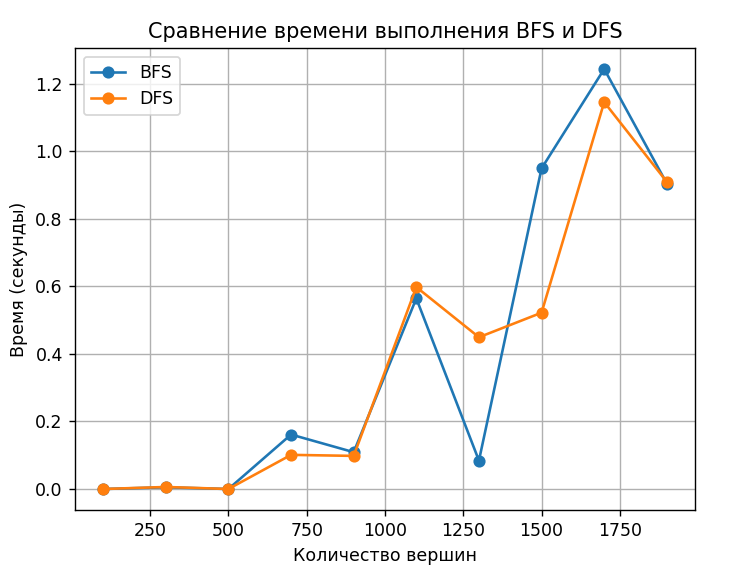
Граф 9:

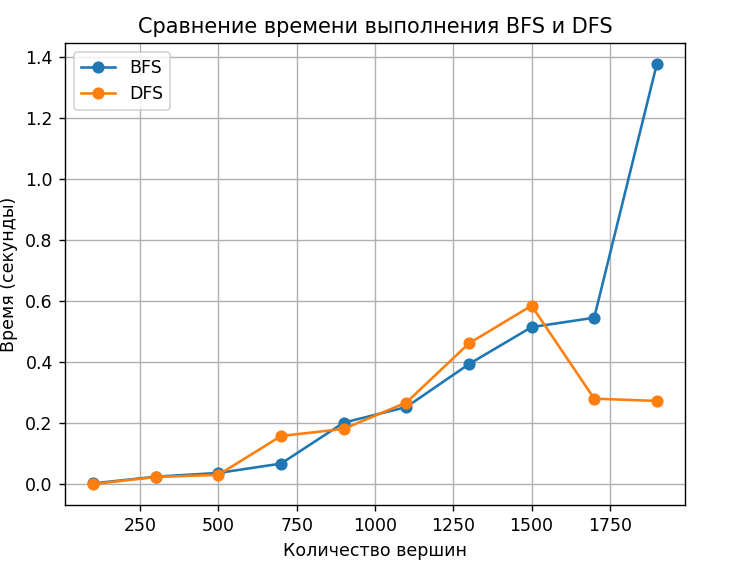
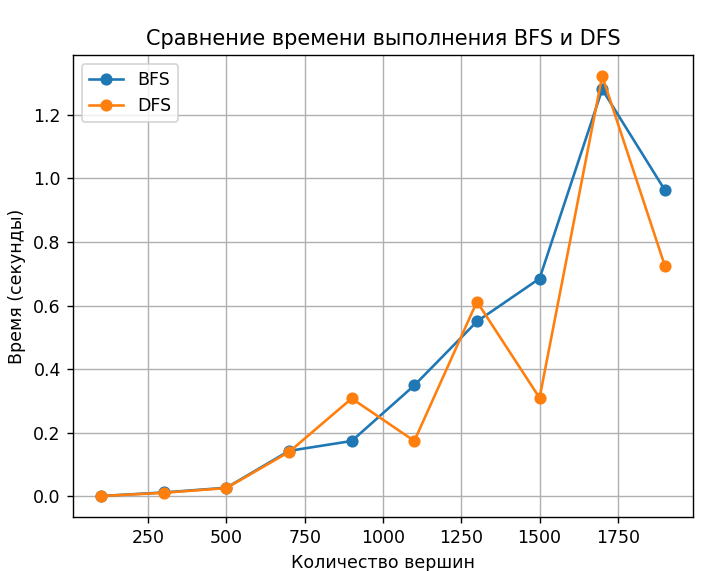
Количество вершин: 1700, Количество ребер: 3300

--------------------------------------------------

Граф 10:

Количество вершин: 1900, Количество ребер: 3700





# Заключение.

Графы это важный инструмент для баз данных, который может быть реализован для многих задач.