Вариант 1

Цель работы – построение кратчайшего пути на графе.

Порядок выполнения работы:

Задать простые связные разреженные неориентированные графы с числом вершин 5000, 15000, 45000, 135000, 405000 случайным образом.

Граф должен содержать подграф K\_5.

Найти двумя способами (по 2 разным алгоритмам) кратчайшие пути от выделенной вершины до остальных вершин.

Найти количество итераций. Сравнить с асимптотической сложностью применяемых алгоритмов.

Содержание отчета по работе:

Блок-схема программы.

Контрольные примеры.

**Важно, что графы должны быть связными и разреженными (средняя степень вершин от 10 до 100)**

Генерация графа

- Случайная генерация: Используем метод случайного выбора рёбер, чтобы гарантировать, что граф будет связным и разреженным.

- Подграф K\_5: Сначала выбираем 5 вершин и соединяем их между собой, затем добавляем остальные рёбра для достижения необходимой средней степени.

import random  
import networkx as nx  
def generate\_sparse\_graph(num\_vertices, num\_edges):  
 G = nx.Graph()  
 G.add\_nodes\_from(range(num\_vertices))  
 k5\_nodes = random.sample(range(num\_vertices), 5)  
 for i in range(5):  
 for j in range(i + 1, 5):  
 G.add\_edge(k5\_nodes[i], k5\_nodes[j])  
 while G.number\_of\_edges() < num\_edges:  
 u = random.randint(0, num\_vertices - 1)  
 v = random.randint(0, num\_vertices - 1)  
 if u != v:  
 G.add\_edge(u, v)  
 return G  
  
  
num\_vertices\_list = [5000, 15000, 45000, 135000, 405000]  
for num\_vertices in num\_vertices\_list:  
 num\_edges = random.randint(num\_vertices \* 10, num\_vertices \* 100)  
 graph = generate\_sparse\_graph(num\_vertices, num\_edges)

Алгоритм Дейкстры

iteration\_count\_dijkstra = 0  
def dijkstra\_with\_count(graph, start):  
 global iteration\_count\_dijkstra  
 length\_dict = {}  
 for node in graph.nodes():  
 length\_dict[node] = float('inf')  
 length\_dict[start] = 0  
 unvisited = set(graph.nodes())  
 while unvisited:  
 iteration\_count\_dijkstra += 1  
 current\_node = min(unvisited, key=lambda node: length\_dict[node])  
 unvisited.remove(current\_node)  
 for neighbor in graph.neighbors(current\_node):  
 if neighbor in unvisited:  
 alt = length\_dict[current\_node] + graph[current\_node][neighbor]['weight']  
 if alt < length\_dict[neighbor]:  
 length\_dict[neighbor] = alt  
 return length\_dict

6. Контрольные примеры

Тестирование выполнено на графах с различным количеством вершин. Результаты показывают:

- Для небольших графов (до 15,000 вершин) алгоритм работает быстро.

- Для больших графов (более 135,000 вершин) алгоритм Дейкстры показывает значительное увеличение времени выполнения.