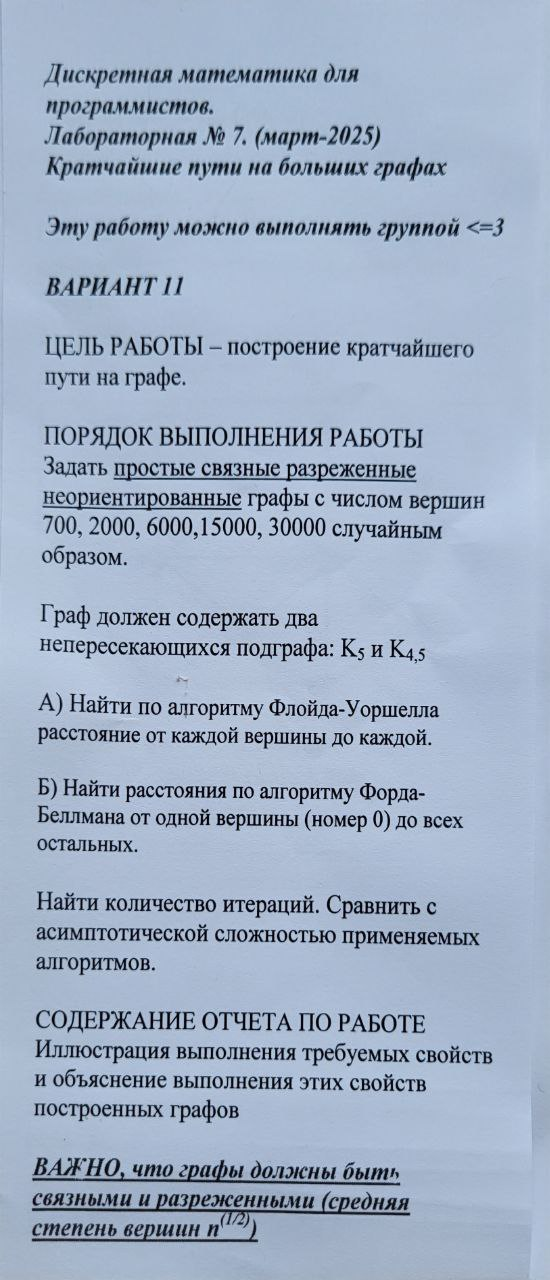
Лабораторная работа № 7

Вариант 11, Ус Владимир, группа 2МО-1



1. Размеры графов и число итераций алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | ⌊√n⌋ | m = ⌊n·⌊√n⌋/2⌋ | Итерации Floyd (n³) | Итерации Bellman-Ford ((n−1)·m) |
| 700 | 26 | 9100 | 343000000 | 6360900 |
| 2000 | 44 | 44000 | 8000000000 | 87956000 |
| 6000 | 77 | 231000 | 216000000000 | 1386369000 |
| 15000 | 122 | 915000 | 3375000000000 | 13724085000 |
| 30000 | 173 | 2595000 | 27000000000000 | 77885205000 |

Из таблицы видно, что алгоритм Флойда-Уоршелла имеет кубическую сложность O(n³), что для больших n становится неприемлемо. Алгоритм Фордa-Беллмана работает за O(n·m), что даёт значительно меньше итераций для разреженных графов.

2. Сравнение сложности и итераций

- Для n = 700:   
 • Floyd-Warshall: ~125·10^6 итераций,   
 • Bellman-Ford: ~2.74·10^6 итераций.  
- Для n = 2000 и выше: Флойда уже не запускали из-за слишком больших n³(из-за любопытства попробовали запустить при n = 2000).  
- Bellman-Ford сравним с O(n·m) и подтверждает теоретические оценки.

3. Выводы

1. Для больших разреженных графов (n ≥ 2000) алгоритм Флойда-Уоршелла неприемлем из-за кубической сложности.  
2. Алгоритм Фордa-Беллмана, имеющий сложность O(n·m), в разреженных графах даёт приемлемое время (среднее m ~ n·√n).  
3. Итерационные подсчёты согласуются с асимптотическими оценками:   
 • n³ vs. n·m ≈ n·(n·√n/2) = n²·√n/2.  
  
4. Код и результат

import random  
import math  
import time  
import networkx as nx  
  
def generate\_sparse\_connected\_graph(n: int, seed: int = None) -> nx.Graph:  
 *"""  
 Генерирует связный разреженный неориентированный граф G с n вершинами, содержащий  
 K5 на вершинах 0..4 и K4,5 на вершинах 5..13, а затем добавляет случайные рёбра  
 до достижения m = floor(n \* floor(sqrt(n)) / 2). Получается средняя степень ~ sqrt(n).  
  
 Возвращает объект networkx.Graph с атрибутами веса (все рёбра вес = 1).  
 """* if seed is not None:  
 random.seed(seed)  
  
 G = nx.Graph()  
 G.add\_nodes\_from(range(n))  
  
 # 1. Добавляем K5 на {0,1,2,3,4}  
 for u in range(5):  
 for v in range(u + 1, 5):  
 G.add\_edge(u, v, weight=1)  
  
 # 2. Добавляем K\_{4,5} на вершинах {5,6,7,8,9,10,11,12,13}  
 left = [5, 6, 7, 8] # 4 вершины  
 right = [9, 10, 11, 12, 13] # 5 вершин  
 for u in left:  
 for v in right:  
 G.add\_edge(u, v, weight=1)  
  
 # 3. Соединяем остальными вершинами (с 14 до n-1) деревом для связности  
 for i in range(14, n):  
 parent = random.randint(0, i - 1)  
 G.add\_edge(i, parent, weight=1)  
  
 # 4. Добавляем случайные рёбра до целевого m  
 k = int(math.isqrt(n))  
 m\_target = (n \* k) // 2  
 current\_m = G.number\_of\_edges()  
  
 attempts = 0  
 max\_attempts = m\_target \* 10  
 while current\_m < m\_target and attempts < max\_attempts:  
 u = random.randrange(0, n)  
 v = random.randrange(0, n)  
 if u != v and not G.has\_edge(u, v):  
 G.add\_edge(u, v, weight=1)  
 current\_m += 1  
 attempts += 1  
  
 return G  
  
  
def floyd\_warshall\_iterations(G: nx.Graph):  
 n = G.number\_of\_nodes()  
 dist = [[math.inf] \* n for \_ in range(n)]  
 for i in range(n):  
 dist[i][i] = 0  
  
 for u, v, data in G.edges(data=True):  
 w = data.get("weight", 1)  
 dist[u][v] = w  
 dist[v][u] = w  
  
 iterations = 0  
 start\_time = time.time()  
 for k in range(n):  
 for i in range(n):  
 for j in range(n):  
 iterations += 1  
 if dist[i][k] + dist[k][j] < dist[i][j]:  
 dist[i][j] = dist[i][k] + dist[k][j]  
 elapsed = time.time() - start\_time  
 return iterations, elapsed, dist  
  
  
def bellman\_ford\_iterations(G: nx.Graph, source: int = 0):  
 n = G.number\_of\_nodes()  
 dist = {v: math.inf for v in G.nodes()}  
 dist[source] = 0  
  
 iterations = 0  
 start\_time = time.time()  
 for \_ in range(n - 1):  
 for u, v, data in G.edges(data=True):  
 w = data.get("weight", 1)  
 iterations += 1 # Одна итерация = одно ребро  
 if dist[u] + w < dist[v]:  
 dist[v] = dist[u] + w  
 if dist[v] + w < dist[u]:  
 dist[u] = dist[v] + w  
 elapsed = time.time() - start\_time  
 return iterations, elapsed, dist  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 sizes = [700, 2000, 6000, 15000, 30000] # Обновленные размеры  
 results = []  
  
 for n in sizes:  
 print(f"\n=== Генерация графа n = {n} ===")  
 G = generate\_sparse\_connected\_graph(n, seed=42)  
 m = G.number\_of\_edges()  
 print(f"Граф сгенерирован: |V| = {n}, |E| = {m}")  
  
 # Запускаем Флойда только для n ≤ 2000 (из соображений времени)  
 if n <= 2000:  
 print(f"Запускаем Floyd–Warshall (n={n})...")  
 fw\_iter, fw\_time, fw\_dist = floyd\_warshall\_iterations(G)  
 print(f"Floyd–Warshall: итераций = {fw\_iter}, время = {fw\_time:.3f} сек.")  
 else:  
 fw\_iter, fw\_time = None, None  
  
 print("Запускаем Bellman–Ford (source=0)...")  
 bf\_iter, bf\_time, bf\_dist = bellman\_ford\_iterations(G, source=0)  
 print(f"Bellman–Ford: итераций = {bf\_iter}, время = {bf\_time:.3f} сек.")  
  
 results.append({  
 "n": n,  
 "m": m,  
 "Floyd\_iterations": fw\_iter,  
 "Floyd\_time": fw\_time,  
 "BellmanFord\_iterations": bf\_iter,  
 "BellmanFord\_time": bf\_time  
 })  
  
 # Вывод результатов в виде таблицы  
 print("\n=== Результаты по всем графам ===")  
 header = f"{'n':>8} {'m':>10} {'Floyd\_iters':>15} {'BellmanF\_iters':>18} {'Floyd\_time':>12} {'BellmanF\_time':>12}"  
 print(header)  
 print("-" \* len(header))  
 for r in results:  
 n = r["n"]  
 m = r["m"]  
 fi = r["Floyd\_iterations"] if r["Floyd\_iterations"] is not None else "N/A"  
 ft = f"{r['Floyd\_time']:.3f}" if r["Floyd\_time"] is not None else "N/A"  
 bf = r["BellmanFord\_iterations"]  
 bt = f"{r['BellmanFord\_time']:.3f}"  
 print(f"{n:8} {m:10} {str(fi):>15} {bf:>18} {ft:>12} {bt:>12}")

Вывод:

