Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

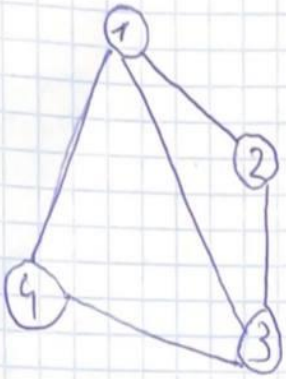
Лабораторна робота № 2
Паралельні та розподілені процеси

Виконав:
студент групи ПМА-32
Шеремета Владислав

Львів-2024

1.1) Побудова графа з чотирма вершинами та п’ятьма ребрами:

Побудуємо граф з 4-ма вершинами і 5-ма ребрами



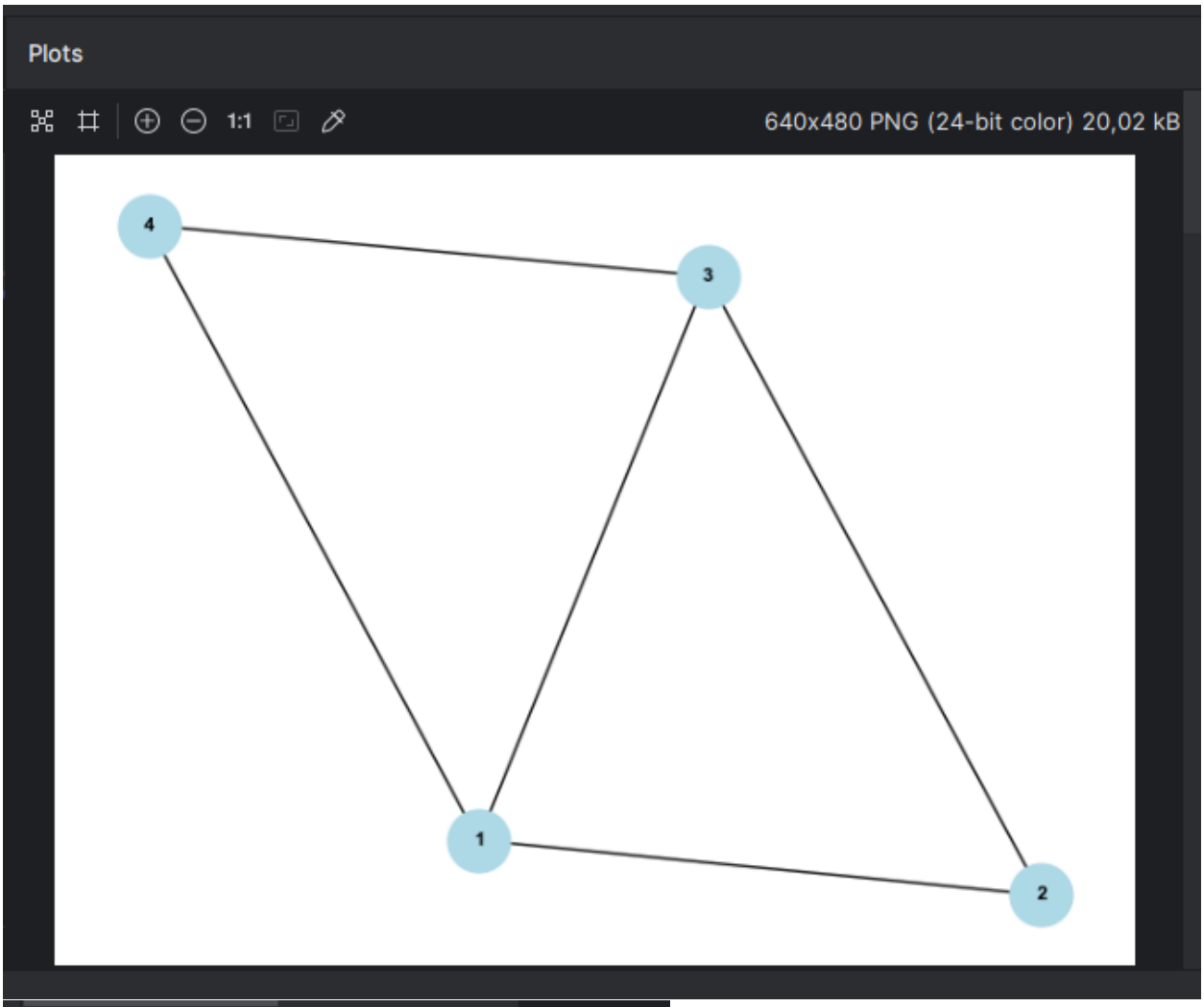
Матриця суміжності:

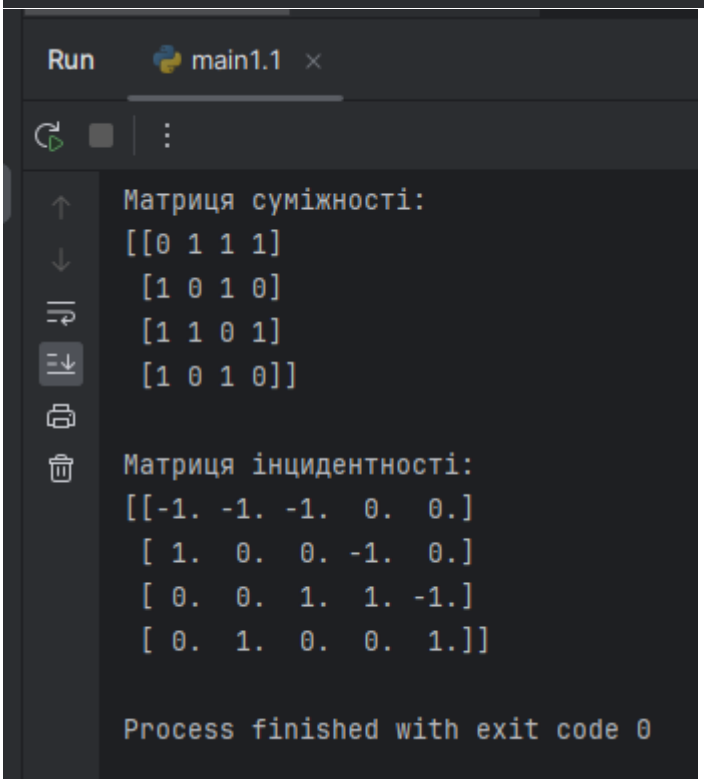
$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Матриця інцидентності:

$$\begin{pmatrix} -1 & -1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

1.2) Вихідні результати виконані програмним кодом:



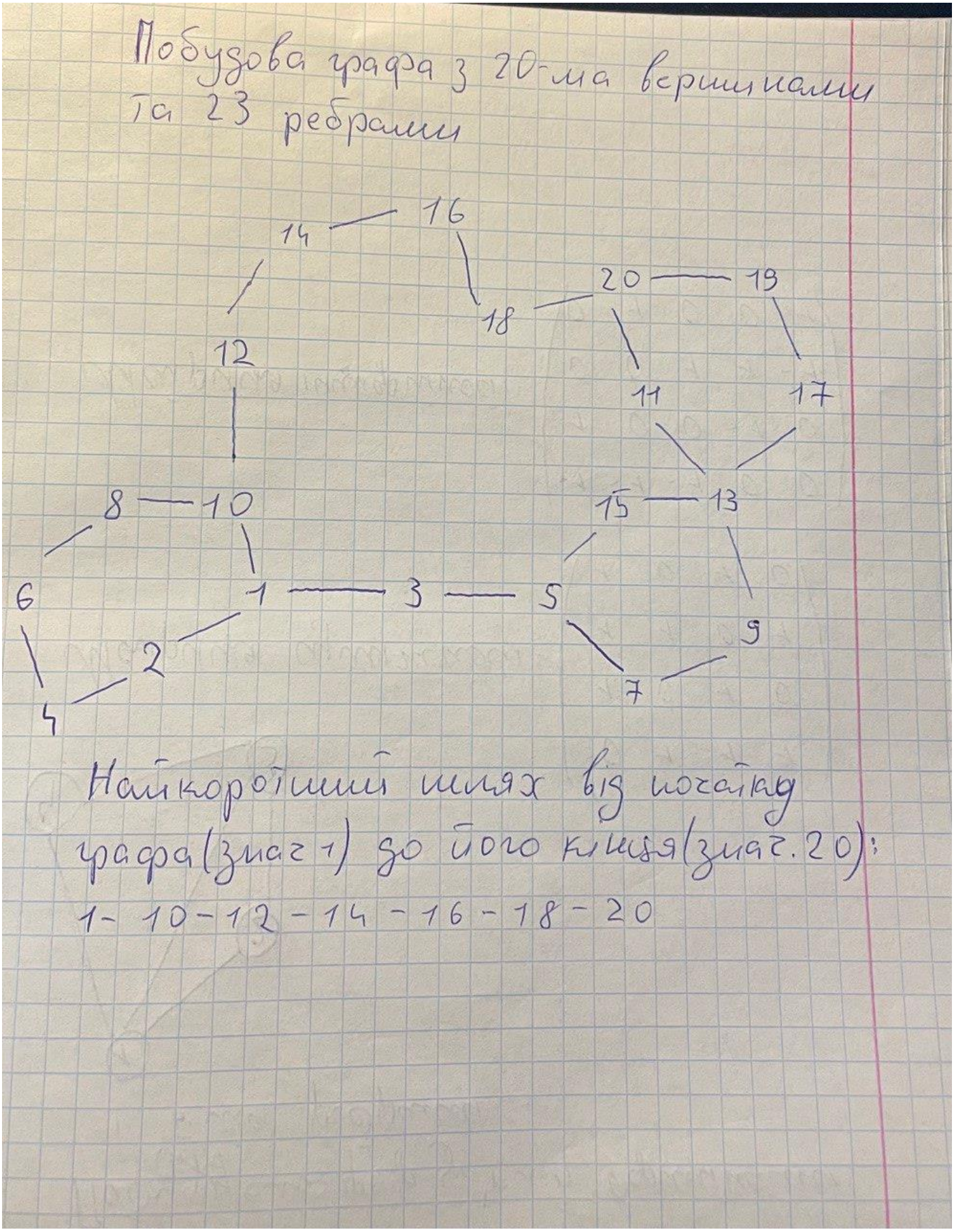


1.3) Програмний код:

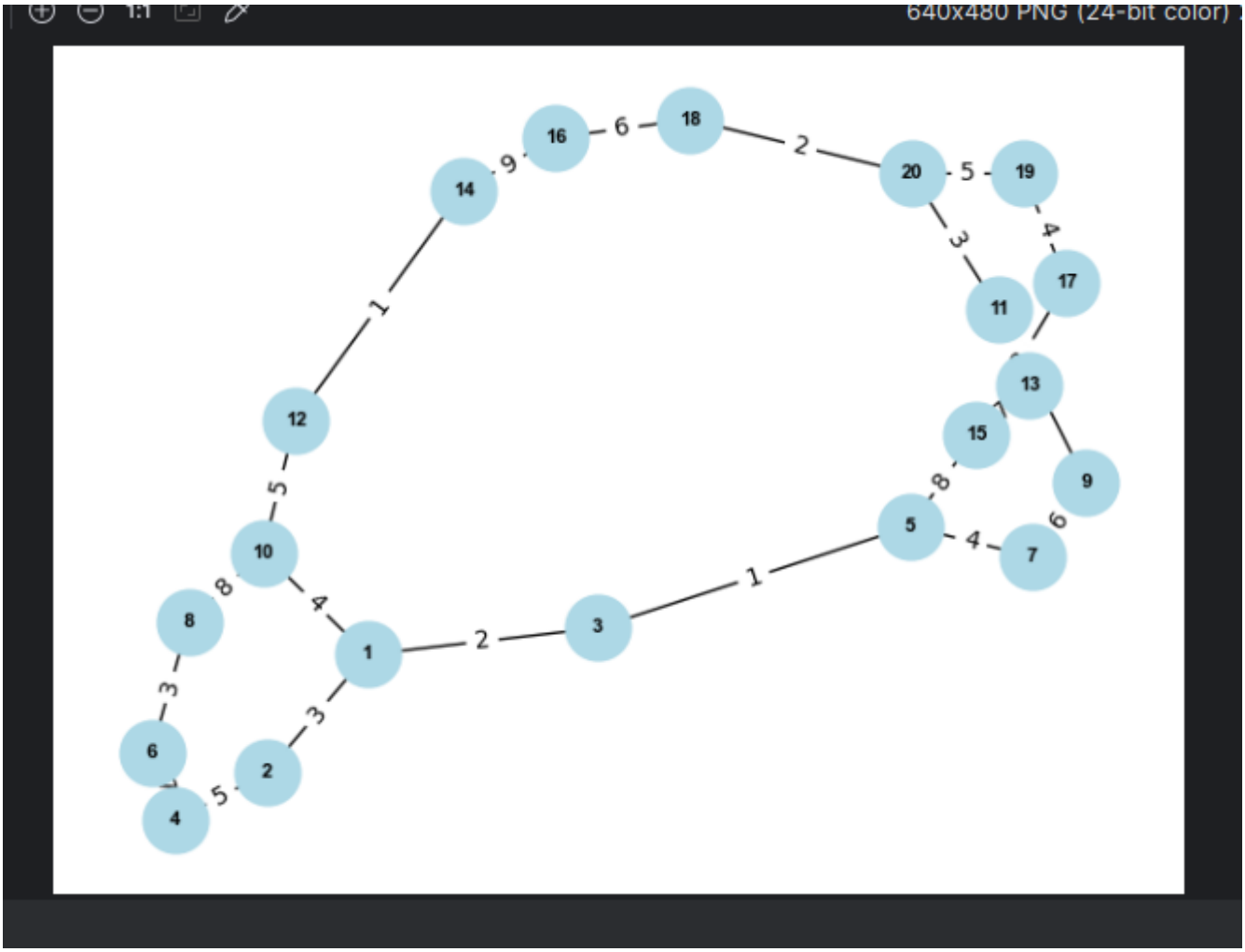
main1.2.py main1.1.py

```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Створення графа
G = nx.Graph()

# Додавання вершин та ребер
G.add_nodes_from(range(1, 5))
edges = [(1, 2), (2, 3), (3, 4), (4, 1), (1, 3)]

G.add_edges_from(edges)

# Виведення матриці суміжності
adjacency_matrix = nx.adjacency_matrix(G).todense()
print("Матриця суміжності:")
print(adjacency_matrix)

# Виведення матриці інцидентності
incidence_matrix = nx.incidence_matrix(G, oriented=True).todense()
print("\nМатриця інцидентності:")
print(incidence_matrix)

# Виведення графа
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=700, node_color='lightblue', font_size=8, font_color='black', font_family='arial')
plt.title("Граф")
plt.show()
```

2.1) Побудова графа з двадцятьма вершинами та двадцятьма п’ятьма ребрами:



2.2) Вихідні результати виконані програмним кодом:



2.3) Програмний код:

```python
import networkx as nx
import matplotlib.pyplot as plt

1 usage
def dijkstra(graph, start):
    # Ініціалізація відстаней та відстаней до кожної вершини
    distances = {vertex: float('infinity') for vertex in graph}
    distances[start] = 0

    # Створення пріоритетної черги для вершин та їх відстаней
    priority_queue = [(0, start)]

    while priority_queue:
        # Вибір вершини з найменшою відстанню
        current_distance, current_vertex = priority_queue.pop(0)

        # Оновлення відстаней до сусідів поточної вершини
        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight['weight']  # Отримання ваги ребра
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                priority_queue.append((distance, neighbor))
                priority_queue.sort()

    return distances

# Створення графа
G = nx.Graph()

# Додавання вершин та ребер
G.add_nodes_from(range(1, 21))
edges = [(1, 2, {'weight': 3}), (1, 3, {'weight': 2}), (2, 4, {'weight': 5}),
         (3, 5, {'weight': 1}), (4, 6, {'weight': 7}), (5, 7, {'weight': 4}),
         (6, 8, {'weight': 3}), (7, 9, {'weight': 6}), (8, 10, {'weight': 8}),
         (9, 11, {'weight': 2}), (10, 12, {'weight': 5}), (11, 13, {'weight': 4}),
         (12, 14, {'weight': 1}), (13, 15, {'weight': 7}), (14, 16, {'weight': 9}),
         (15, 17, {'weight': 3}), (16, 18, {'weight': 6}), (17, 19, {'weight': 4}),
         (18, 20, {'weight': 2}), (19, 20, {'weight': 5}), (1, 10, {'weight': 4}),
         (5, 15, {'weight': 8}), (11, 20, {'weight': 3})]

G.add_edges_from(edges)

# Задання початкової вершини для алгоритму Дейкстри
start_node = 1

# Виклик алгоритму Дейкстри
shortest_paths = dijkstra(G, start_node)

# Виведення результатів
print("Найкоротші шляхи від вершини {}:".format(start_node))
for vertex, distance in shortest_paths.items():
    print("Вершина {}: Відстань - {}".format( *args: vertex, distance))

# Виведення графа
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_size=700, node_color='lightblue', font_size=8, font_color='black', font_family='arial')
labels = nx.get_edge_attributes(G, name: 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
```

Алгоритм Дейкстри - це алгоритм пошуку найкоротших шляхів в графі з невід'ємними вагами ребер. Він був розроблений голландським математиком Едсгером Дейкстрою в 1956 році.

Основна ідея алгоритму полягає в тому, щоб поступово визначати найкоротші відстані від початкового вузла (вершини) до всіх інших вузлів графа. Алгоритм підтримує масив відстаней, який оновлюється, коли знаходиться коротший шлях.

Основні етапи алгоритму Дейкстри:

1. Ініціалізація відстаней до всіх вузлів, крім початкового, як нескінченно великі.
2. Позначення відстаней до початкового вузла як 0.
3. Вибір початкового вузла та оновлення відстаней до його сусідів, якщо знайдено коротший шлях.
4. Повторення кроку 3 для всіх вузлів графа.
5. Кінцевий результат - масив найкоротших відстаней від початкового вузла до всіх інших.
6. Алгоритм Дейкстри ефективний для графів з невід'ємними вагами ребер і використовується в багатьох областях, таких як мережеве проектування, транспортна логістика та інші задачі, пов'язані з оптимізацією шляхів.

Висновок:

За допомогою програмного забезпечення обчислив матриці суміжності та інцидентності, а також найкоротший шлях від початку графа до його кінця. Отримані результати перевірив, обчисливши це вручну, та зрозумів, що відповіді збігаються.