ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

Паралельні та розподілені обчислення
ЛАБОРАТОРНА РОБОТА №5

Тема: «Алгоритм Флойда».

Виконала:
Ст. *Пелещак Вероніка*
ПМІ-35с

2025

Завдання: Для орієнтованого зваженого графа G(V,F), де V={a0, a1,…an} – множина вершин (n –велике число), а F множина орієнтованих ребер (шляхів) між вершинами, використовуючи алгоритм Флойда, знайти найкоротший шлях між заданими вузлами a та b. Для різної розмірності графів та довільних вузлів a та b порахувати час виконання програми без потоків та при заданих k потоках розпаралелення.

Опис програми:

1. Генерація графа (*generateGraph*):

- Створює випадковий орієнтований граф n x n.
- Відсутні ребра мають вагу INF (велике число).
- Всі ребра отримують випадкові ваги в діапазоні, який можна задати (цілі або дробові).
- Головна діагональ = 0 (відстань від вершини до самої себе).

2. Послідовний алгоритм Флойда (*floydSequential*):

- Обчислює найкоротші шляхи між усіма парами вершин.
- Три вкладені цикли: k — проміжна вершина; i та j — початкова і кінцева вершини.
- Оновлює відстань, якщо знайдено коротший шлях через k.

3. Паралельний алгоритм Флойда (*floydParallel*):

- Використовує k потоків для паралельного обчислення рядків матриці відстаней.
- Кожен потік обробляє певну частину рядків, що пришвидшує роботу на великих графах.

- Після обчислення кожного проміжного кроку потоки об’єднуються (**join**).

4. Основна програма (*main*):

- Користувач вводить:
  - Кількість вершин n;
  - Вершини a і b для пошуку найкоротшого шляху;
  - Кількість потоків k для паралельної версії.
- Створюється граф та копії для послідовної і паралельної версій.
- Виконується алгоритм Флойда у послідовному та паралельному варіантах.
- Виводиться:
  - Час виконання обох версій;
  - Найкоротша відстань між вершинами a і b;
  - Speedup і Efficiency для паралельного виконання.

5. Ключові моменти:

- Використання власної функції генерації графа з випадковими вагами.
- Використання потоків (**std::thread**) для розподілу обчислень у паралельній версії.
- Вимірювання часу виконання через **chrono::high_resolution_clock**.
- Обчислення **Speedup** та **Efficiency** для оцінки ефективності паралельного алгоритму.

Результати:

```
Введіть кількість вершин графа: 500
Введіть номери вершин a і b (0..499): 0 499
Введіть кількість потоків: 4

Послідовна версія: 1384 мс
Паралельна версія (4 потоків): 448 мс

Найкоротша відстань між вершинами 0 і 499: 82.5439
Speedup: 3.08929
Efficiency: 0.772321

Process finished with exit code 0
```

```
Введіть кількість вершин графа: 700
Введіть номери вершин a і b (0..699): 12 530
Введіть кількість потоків: 6

Послідовна версія: 3749 мс
Паралельна версія (6 потоків): 847 мс

Найкоротша відстань між вершинами 12 і 530: 100.425
Speedup: 4.42621
Efficiency: 0.737702

Process finished with exit code 0
```

```
Введіть кількість вершин графа: 1000
Введіть номери вершин a і b (0..999): 3 997
Введіть кількість потоків: 8

Послідовна версія: 10825 мс
Паралельна версія (8 потоків): 2630 мс

Найкоротша відстань між вершинами 3 і 997: 117.806
Speedup: 4.11597
Efficiency: 0.514496

Process finished with exit code 0
```

```
Введіть кількість вершин графа: 1500
Введіть номери вершин a і b (0..1499): 28 1402
Введіть кількість потоків: 12

Послідовна версія: 36595 мс
Паралельна версія (12 потоків): 8617 мс

Найкоротша відстань між вершинами 28 і 1402: 51.3327
Speedup: 4.24684
Efficiency: 0.353903

Process finished with exit code 0
```