ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра програмування

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №6

Тема: «Алгоритм Дейкстри»

Виконав:

Ст. *Лук'янчук Денис*

Група ПМІ-33

2025

Тема: «Алгоритм Дейкстри»

Мета роботи: Ознайомитися з принципами багатопоточності та синхронізації потоків на прикладі алгоритму Дейкстри. Навчитися вимірювати час виконання послідовних і паралельних обчислень, оцінювати прискорення та ефективність роботи алгоритму, а також дослідити вплив кількості потоків на продуктивність пошуку найкоротших шляхів у графі.Теоретичний матеріал

Орієнтований зважений граф $G(V, F)$складається з:

- **V = {a0, a1, …, an}** – множина вершин,
- **F** – множина орієнтованих ребер, де кожне ребро має вагу $w(i, j)$.

Вага ребра $w(i, j)$показує «довжину» або «вартість» переходу від вершини i до вершини j. Якщо ребро відсутнє, його вага приймається як нескінченність (∞).

Алгоритм Дейкстри дозволяє знайти найкоротший шлях від однієї вершини до всіх інших у графі без від'ємних ваг. Основна ідея полягає у послідовному розширенні множини вершин, для яких вже знайдено мінімальну відстань. На кожному кроці обирається вершина u з найменшою відстанню dist[u], після чого оновлюються відстані до всіх суміжних вершин v:

$$dist[v] = \min(dist[v], dist[u] + w(u, v))$$

Складність послідовної версії алгоритму: $O(V^2)$ (без пріоритетної черги).

Паралельна реалізація через потоки:

- Матриця відстаней розбивається на частини, кожен потік обробляє свою частину рядків.
- Після завершення обробки всіх потоків відбувається синхронізація (Join()).
- Прискорення та ефективність визначаються формулами:

$$S_k = \frac{T_{\text{посл}}}{T_{\text{парал}}}, E_k = \frac{S_k}{k} \times 100\%$$

де k – кількість потоків.

Хід роботи

Під час виконання роботи я реалізував алгоритм Дейкстри двома способами: послідовно та паралельно через потоки.

Спершу я згенерував граф випадковим чином, причому ваги ребер приймали значення від 5 до 100, а відсутні ребра позначалися нескінченністю (∞). Для експериментів було обрано три розміри графів: n = 100, 1000, 10000, а також різну кількість потоків: k = 4, 8, 16. Для перевірки коректності результатів обрані конкретні вершини a (початкова) та b (кінцева), між якими обчислювався найкоротший шлях.

У послідовній версії алгоритму відстані від заданої вершини обчислювалися по черзі: обиралася вершина з мінімальною відстанню, і оновлювалися значення відстаней до суміжних вершин.

У паралельній версії масив відстаней розбивався на блоки, і кожен потік обробляв свій блок вершин одночасно. Після завершення роботи всіх потоків відбувалася синхронізація (Join()). Незважаючи на паралельну обробку, загальний вибір наступної вершини з мінімальною відстанню залишався послідовним, що обмежувало ефективність паралельного виконання.

Для оцінки продуктивності я вимірював час виконання обох версій алгоритму та розраховував прискорення і ефективність. Коректність роботи перевірялася шляхом порівняння результатів послідовної та паралельної версій для конкретних вершин.

Результати експерименту

Граф n = 100:

```
Граф 100x100
Пошук шляху: a=10, b=80

Послідовно: 0,64 мс
Найкоротша відстань від 10 до 80: 76

Паралельно (4 потоків): 38,28 мс
Найкоротша відстань від 10 до 80: 76

Показники продуктивності
Прискорення (S4) = 0,017
Ефективність (E4) = 0,42%
```

```
Граф 100x100
Пошук шляху: a=10, b=80

Послідовно: 0,54 мс
Найкоротша відстань від 10 до 80: 76

Паралельно (8 потоків): 70,58 мс
Найкоротша відстань від 10 до 80: 76

Показники продуктивності
Прискорення (S8) = 0,008
Ефективність (E8) = 0,10%
```

```
Граф 100x100
Пошук шляху: a=10, b=80

Послідовно: 0,54 мс
Найкоротша відстань від 10 до 80: 76

Паралельно (16 потоків): 136,52 мс
Найкоротша відстань від 10 до 80: 76

Показники продуктивності
Прискорення (S16) = 0,004
Ефективність (E16) = 0,02%
```

Послідовне виконання працює швидше, ніж паралельне через накладні витрати на створення потоків і блокування даних. Прискорення практично дорівнює нулю, ефективність падає при збільшенні k.

Граф n = 1000:

```
Граф 1000x1000
Пошук шляху: a=10, b=753

Послідовно: 10,28 мс
Найкоротша відстань від 10 до 753: 25

Паралельно (4 потоків): 490,63 мс
Найкоротша відстань від 10 до 753: 25

Показники продуктивності
Прискорення (S4) = 0,021
Ефективність (E4) = 0,52%
```

```
Граф 1000x1000
Пошук шляху: a=10, b=753

Послідовно: 11,65 мс
Найкоротша відстань від 10 до 753: 25

Паралельно (8 потоків): 784,24 мс
Найкоротша відстань від 10 до 753: 25

Показники продуктивності
Прискорення (S8) = 0,015
Ефективність (E8) = 0,19%
```

```
Граф 1000x1000
Пошук шляху: a=10, b=753

Послідовно: 10,39 мс
Найкоротша відстань від 10 до 753: 25

Паралельно (16 потоків): 1626,53 мс
Найкоротша відстань від 10 до 753: 25

Показники продуктивності
Прискорення (S16) = 0,006
Ефективність (E16) = 0,04%
```

Послідовний алгоритм залишається набагато швидшим. Паралельний варіант зі зростанням потоків демонструє ще більші накладні витрати, прискорення та ефективність продовжують падати.

Граф n = 10000:

```
Граф 10000x10000
Пошук шляху: a=378, b=7531

Послідовно: 906,67 мс
Найкоротша відстань від 378 до 7531: 15

Паралельно (4 потоків): 4609,11 мс
Найкоротша відстань від 378 до 7531: 15

Показники продуктивності
Прискорення (S4) = 0,197
Ефективність (E4) = 4,92%
```

```
Граф 10000x10000
Пошук шляху: a=378, b=7531

Послідовно: 942,89 мс
Найкоротша відстань від 378 до 7531: 15

Паралельно (8 потоків): 8440,31 мс
Найкоротша відстань від 378 до 7531: 15

Показники продуктивності
Прискорення (S8) = 0,112
Ефективність (E8) = 1,40%
```

```
Граф 10000x10000
Пошук шляху: a=378, b=7531

Послідовно: 906,80 мс
Найкоротша відстань від 378 до 7531: 15

Паралельно (16 потоків): 15969,96 мс
Найкоротша відстань від 378 до 7531: 15

Показники продуктивності
Прискорення (S16) = 0,057
Ефективність (E16) = 0,35%
```

Паралельна обробка через потоки показала мінімальне збільшення ефективності порівняно з попередніми прикладами. Прискорення сильно менше 1, швидкість виконання надалі сильно повільніша за послідовний.

Граф n = 25000:

Послідовний алгоритм Дейкстри є набагато швидшим ніж паралельний навіть при великій кількості вершин графа. Паралельний варіант зі зростанням потоків демонструє ще більші накладні витрати, прискорення та ефективність продовжують падати при збільшені кількості потоків. Но чим більше кількість вершин у графі тим ефективність стає більша у великому графі в порівнянні з малим.

```
Граф 25000x25000
Пошук шляху: a=1240, b=11260

Послідовно: 17486,62 мс
Найкоротша відстань від 1240 до 11260: 13

Паралельно (4 потоків): 74686,22 мс
Найкоротша відстань від 1240 до 11260: 13

Показники продуктивності
Прискорення (S4) = 0,234
Ефективність (E4) = 5,85%
```

```
Граф 25000x25000
Пошук шляху: a=1240, b=11260

Послідовно: 6067,21 мс
Найкоротша відстань від 1240 до 11260: 13

Паралельно (8 потоків): 25852,14 мс
Найкоротша відстань від 1240 до 11260: 13

Показники продуктивності
Прискорення (S8) = 0,235
Ефективність (E8) = 2,93%
```

```
Граф 25000x25000
Пошук шляху: a=1240, b=11260

Послідовно: 6308,43 мс
Найкоротша відстань від 1240 до 11260: 13

Паралельно (16 потоків): 42515,56 мс
Найкоротша відстань від 1240 до 11260: 13

Показники продуктивності
Прискорення (S16) = 0,148
Ефективність (E16) = 0,93%
```

Висновки по потоках:

- Збільшення кількості потоків не прискорює виконання, а навпаки — додає накладні витрати.
- Усі експерименти підтверджують, що послідовне виконання є найефективнішим у даній реалізації.
- Прискорення падає зі збільшенням k, ефективність падає пропорційно.

Висновок: У лабораторній роботі я ознайомився з принципами багатопоточності та синхронізації потоків на прикладі алгоритму Дейкстри для знаходження найкоротших шляхів. Було реалізовано послідовне та паралельне обчислення, що дозволило оцінити прискорення та ефективність для графів різного розміру.

Експерименти показали, що **послідовне** виконання швидше за паралельне, а збільшення кількості потоків знижує ефективність через накладні витрати на синхронізацію. Найефективнішим залишається послідовне виконання для графів будь-якого розміру.