Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

Звіт

Лабораторна робота №6

Тема: «Алгоритм Дейкстри»

з дисципліни "Паралельні та розподільні обчислення"

Виконав студент групи ПМі-31
Яцуляк Андрій

Львів 2023 р.

Мета: Написати програми знаходження найкоротшого шляху між заданою вершиною та всіма іншими у зваженому орієнтованому графі, використовуючи алгоритм Дейкстри (послідовний та паралельний).

Теоретичний матеріал

Граф — це структура, що складається з набору об’єктів, у якому деякі пари об’єктів у певному сенсі «пов’язані». Об’єкти відповідають математичним абстракціям, які називаються вершинами, а кожна з пов’язаних пар вершин називається ребром. Як правило, граф зображується у вигляді діаграми як набір точок або кіл для вершин, з’єднаних лініями або кривими для ребер. Графи є одним з об’єктів вивчення дискретної математики.

Графом G = (V, E) називають сукупність двох множин: скінченої непорожньої множини V **вершин** і скінченої множини E **ребер**, які з'єднують пари вершин. Ребра зображаються невпорядкованими парами вершин (u, v).

У графі можуть бути **петлі** — ребра, що починаються і закінчуються в одній вершині, а також повторювані ребра (кратні, або паралельні). Якщо в графі немає петель і кратних ребер, то такий граф називають **простим**. Якщо граф містить кратні ребра, то граф називають **мультиграфом**.

Ребра вважаються неорієнтованими в тому сенсі, що пари (u, v) та (v,u) вважаються одним і тим самим ребром.

Зваженим називають простий граф, кожному ребру e якого приписано дійсне число w(e). Це число називають **вагою** ребра e.

Алгоритм Дейкстри – знаходження найкоротшого шляху від заданої вершини графа до всіх інших вершин цього графа.

Швидкодія O(E * logV).

Об’єм пам’яті O(V).

Хід роботи

Завдання виконав мовою програмування Java у середовищі IntelliJ IDEA. Написав повноцінну програму для роботи з зваженими орієнтованими графами.

Задається граф матрицею інцидентності, де verticesOfGraph[i][j] – вага орієнтованого ребра від i до j.

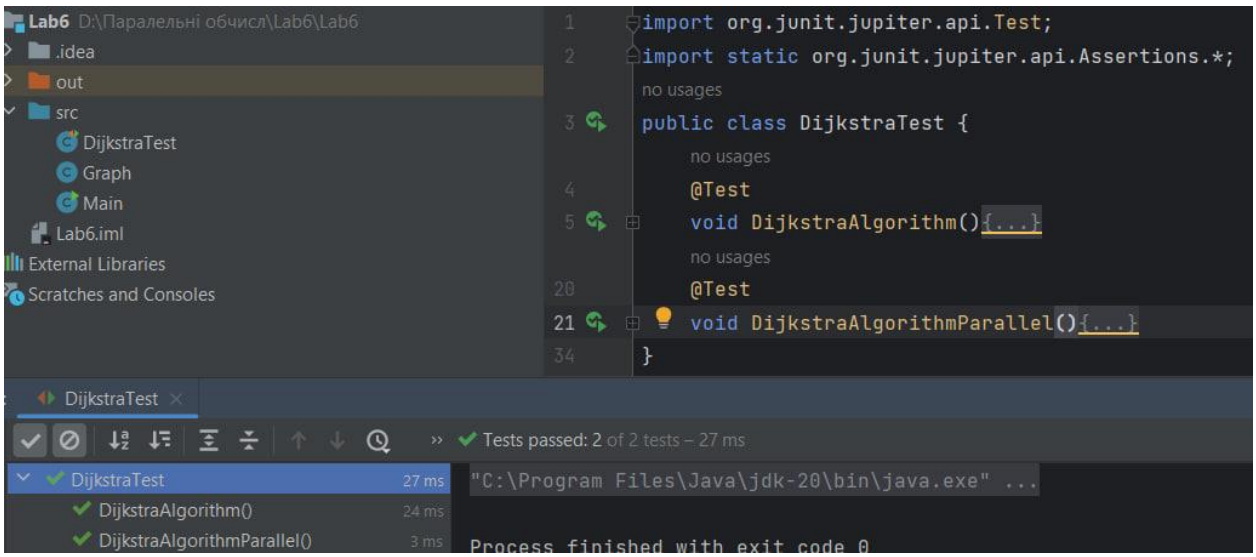
```java
public class Graph {
    17 usages
    private final int numOfVertex;
    13 usages
    private final Integer[][] verticesOfGraph;
    3 usages
    private static int threadsNumber = 1;

    1 usage
    public Graph(int numOfVertex){...}
    2 usages
    public Graph(Integer[][] verticesOfGraph){...}
    1 usage
    public void setEdge(int source, int destination, int weight){...}
    no usages
    public void removeEdge(int source, int destination){...}
    1 usage
    public void fillGraph(int numberOfEdges){...}
    no usages
    public Integer[][] getVertices() { return this.verticesOfGraph; }
    no usages
    public static int getThreadsNumber() { return threadsNumber; }
    1 usage
    public static void setThreadsNumber(int threadsNumber) { Graph.threadsNumber = threadsNumber; }
    2 usages
    public Integer[] DijkstraAlgorithm(int source) {...}
    10 usages
    private static class VertexDistancePair implements Comparable<VertexDistancePair> {...}
```

```java
    2 usages
    public Integer[] DijkstraAlgorithmParallel(int source) {...}
    3 usages
    private class DijkstraTask extends RecursiveTask<Void> {...}
}
```

Робота з графами

Перед початком основної роботи unit тестів, аби перевірити методи на правильність роботи:



Переконавшись, що все працює правильно, створив граф з 1000 вершинами та 100000 орієнтованими ребрами зі значеннями від 1 до 100. Найкоротший шлях я знайшов для 100 вершини. Алгоритм Дейкстри є досить важким для реалізації обчислення паралельно, адже він залежить від попередніх кроків. Тому для обчислення методу паралельно я використав фрейморк ForkJoinPool. Я рекурсивно розділяв обчислення на менші підзавдання, які відбувалися доти, доки підзавдання не включали в себе ту кількість вершин, яку задає користувач з консолі, після чого кожне підзавдання працювало послідовно. Циклом для різної кількості ядер обчислив час послідовної та паралельної роботи алгоритму, прискорення, ефективність:

```
Enter the number of vertices: 1000
Enter the number of edges: 100000
Enter the number of vertex: 100
Sequential time: 22372100 nanoseconds

Threads number: 2
Parallel time: 8693800 nanoseconds
Speed up: 2.573339621339345
Efficiency: 1.2866698106696726

Threads number: 3
Parallel time: 8046400 nanoseconds
Speed up: 2.7803862596937763
Efficiency: 0.9267954198979255

Threads number: 4
Parallel time: 7332600 nanoseconds
Speed up: 3.0510460136922783
Efficiency: 0.7627615034230696

Threads number: 8
Parallel time: 6259600 nanoseconds
Speed up: 3.574046264937057
Efficiency: 0.4467557831171321

Threads number: 16
Parallel time: 6754400 nanoseconds
Speed up: 3.3122261044652377
Efficiency: 0.20701413152907736
```

Отже, найкращого прискорення з показником 3.57 вдалось досягнути при 8-ми потоках, ефективність у 1.28 є найкращою для 2-х потоків. При збільшенні кількості потоків ефективність зменшується.

Висновок. Під час виконання лабораторної роботи я написав програму для знаходження найкоротшого шляху між заданою вершиною та всіма іншими вершинами у зваженому орієнтованому графі, використовуючи алгоритм Дейкстри (послідовний та паралельний), обчислив прискорення та ефективність для різної кількості потоків та навчився аналізувати ці дані.