Parallel Algorithms Implementation

Ivan Pasportnikov

24.12.2023

Contents

1 Introduction

This document presents a semester work focused on implementing two algorithms for working with Graph:

- Dijkstra
- DFS

Additionally, their performance and efficiency, including multi-threaded variants for improved performance, are presented.

All examples will use two identical complex functions for computation and perform computation with a larger number of subintervals for comparison of performance.

2 CLI

For different operating modes of the program, a CLI interface was introduced.

When running the program with an invalid argument or argument –help (-h):

```
Usage: ./sem_work [--help] [--dfs <src>] [--dijkstra <src> <dest>]
    [--PDFS <src>] [--PDKS <src> <dest>] [--exit]
Options:
 --help        Print this help message.
 --dfs <src>   Perform a depth-first search from vertex src.
 --dijkstra <src> <dest> Perform Dijkstra's algorithm from vertex
    src to vertex dest.
 --PDFS <src>  Perform a parallel depth-first search from vertex src.
 --PDKS <src> <dest> Perform parallel Dijkstra's algorithm from
    vertex src to vertex dest.
 --exit        Exit and close the program
```

3 Examples (arguments –dfs,–PDFS,–dijkstra,–PDKS,–help) on simple Graph

3.1 Simple Graph:

```
Graph graph(5);
    graph.addEdge(0, 1, 1);
    graph.addEdge(0, 4, 1);
    graph.addEdge(1, 2, 1);
    graph.addEdge(1, 4, 1);
    graph.addEdge(2, 3, 1);
    graph.addEdge(3, 4, 1);
```

3.2 Using Depth-First Search (DFS) on a simple graph:

Input:

```
--dfs 0
```

Output:

```
Depth-first search from vertex 0:
0 1 2 3 4
Time taken by depth-first search: 0 ms
```

3.3 Using Dijkstra's Algorithm on a simple graph:

Input:

```
--dijkstra 0 4
```

Output:

```
Shortest path from vertex 0 to vertex 4:
0 4
Time taken by Dijkstra's algorithm: 0 ms
```

3.4 Using Parallel Depth-First Search (PDFS) on a simple graph:

Input:

```
--PDFS 0
```

Output:

```
Parallel depth-first search from vertex 0:
0 1 2 3 4
Time taken by parallel depth-first search: 0 ms
```

3.5 Using Parallel Dijkstra's Algorithm (PDKS) on a simple graph:

Input:

```
--PDKS 0 4
```

Output:

```
Parallel Dijkstra Shortest path from vertex 0 to vertex 4:
0 4
Time taken by parallel Dijkstra's algorithm: 0 ms
```

3.6 Error inputs

Input:

```
--invalid
```

Output:

```
Error: Invalid command-line argument.
```

Input:

```
--dfs
```

Output:

```
Error: Missing source vertex for depth-first search.
```

4 Performance Comparison in Different Modes

4.1 Large Graph:

```
Graph graph(100000);
for (int i = 0; i < 100000; ++i) {
    graph.addEdge(i, (i + 1) % 100000, 1); // connectin' each vertex
        to the next one
}
```

4.2 Using DFS on a large graph:

```
Graph size: 100000 vertices
Number of edges: 99999
```

Output:

```
Depth-first search from vertex 0:
0 1 2 3 4 ...
Time taken by depth-first search: 49 ms
```

4.3 Using Parallel DFS on a large graph:

```
Graph size: 100000 vertices
Number of edges: 99999
```

Output:

```
Depth-first search from vertex 0:
0-1 1-2 2-3 3-4 ...
Time taken by depth-first search: 37 ms
```

4.4 Using Dijkstra's Algorithm on a large graph:

```
Graph size: 100000 vertices
Number of edges: 99999
```

Output:

```
Shortest path from vertex 0 to vertex 4:
0 1 2 3 4
Time taken by Dijkstra's algorithm: 51 ms
```

4.5 Using Parallel Dijkstra's Algorithm on a large graph:

```
Graph size: 100000 vertices
Number of edges: 99999
```

Output:

```
Parallel Dijkstra Shortest path from vertex 0 to vertex 4:
0 1 2 3 4
Time taken by parallel Dijkstra's algorithm: 40 ms
```

It is visible that the result on large graphs in parallel mode is faster than the result in classical mode.

All calculations were performed on the Windows 10 operating system using an Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz processor.