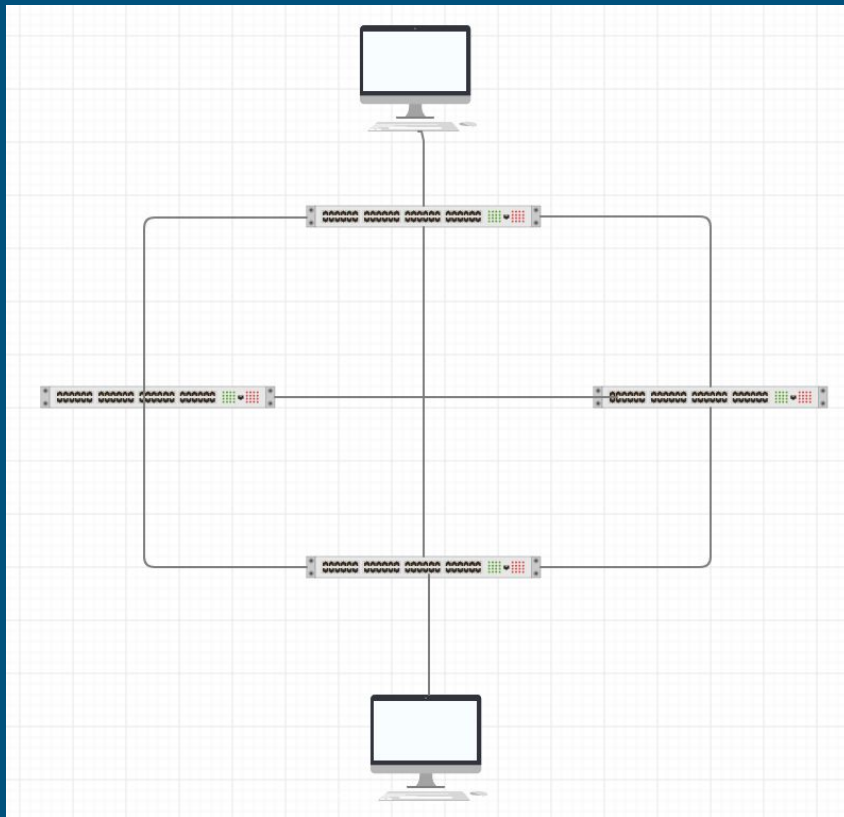


Load Balancer z wykorzystaniem algorytmu Bellmana-Forda

Adrian Kocemba, Bartłomiej Szabat,
Łukasz Matuszyk, Michał Sypek

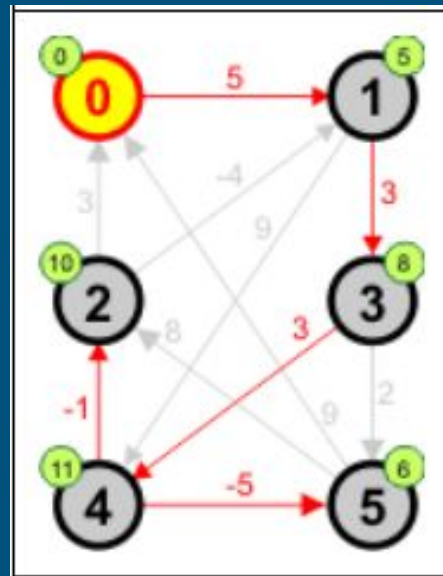
Topologia

Domyślnie każda ze ścieżek
ma wagę równą 1.



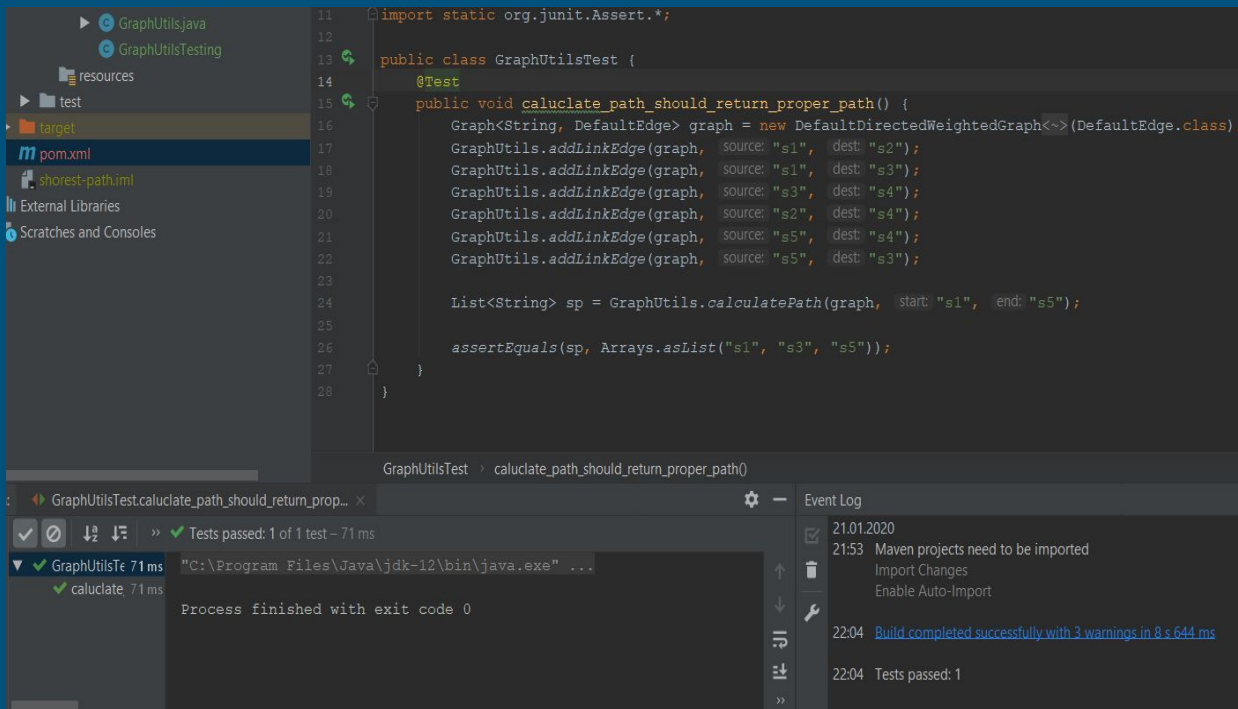
Algorytm Bellmana-Forda

- algorytm do wyszukiwania najkrótszych ścieżek w grafie ważonym z wierzchołka źródłowego do wszystkich pozostałych
- w odróżnieniu od algorytmu Dijkstry, algorytm Bellmana-Forda działa poprawnie także dla grafów z wagami ujemnymi (nie może jednak wystąpić cykl o łącznej ujemnej wadze osiągalny ze źródła)



Weryfikacja zaimplementowanego algorytmu

Testy jednostkowe
sprawdzające
poprawność
działania algorytmu



The screenshot displays an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'test' directory with a 'target' subdirectory. The code editor shows a Java file named 'GraphUtilsTest.java' with the following content:

```
11 import static org.junit.Assert.*;
12
13 public class GraphUtilsTest {
14     @Test
15     public void calculate_path_should_return_proper_path() {
16         Graph<String, DefaultEdge> graph = new DefaultDirectedWeightedGraph<>(DefaultEdge.class);
17         GraphUtils.addLinkEdge(graph, source: "s1", dest: "s2");
18         GraphUtils.addLinkEdge(graph, source: "s1", dest: "s3");
19         GraphUtils.addLinkEdge(graph, source: "s3", dest: "s4");
20         GraphUtils.addLinkEdge(graph, source: "s2", dest: "s4");
21         GraphUtils.addLinkEdge(graph, source: "s5", dest: "s4");
22         GraphUtils.addLinkEdge(graph, source: "s5", dest: "s3");
23
24         List<String> sp = GraphUtils.calculatePath(graph, start: "s1", end: "s5");
25
26         assertEquals(sp, Arrays.asList("s1", "s3", "s5"));
27     }
28 }
```

Below the code editor, the IDE shows the test results for the 'calculate_path_should_return_proper_path' test. The test passed successfully, taking 71 ms. The output of the test is displayed in the console:

```
Process finished with exit code 0
```

The Event Log on the right side of the IDE shows the following messages:

- 21:01:2020
- 21:53 Maven projects need to be imported
- Import Changes
- Enable Auto-Import
- 22:04 Build completed successfully with 3 warnings in 8 s 644 ms
- 22:04 Tests passed: 1

Dokumentacja projektu

Dokumentacja projektu:

<https://github.com/bartegu/ssp>

README.md

Sieci Sterowane Programowo - Projekt

Load balancer z wykorzystaniem algorytmu Bellmana-Forda

Funkcjonalność została dodana jako nowy moduł kontrolera Floodlight.

Cel

Celem projektu było dynamiczne wybieranie ścieżki dla przychodzącego flow. Osiągnięte zostało to z wykorzystaniem algorytmu Bellmana-Forda. Dla każdego przepływu optymalna trasa to ta wyliczona przez algorytm (suma wag ścieżek). Po przepuszczeniu flow daną trasą jej waga jest zwiększana. Dla kolejnego przepływu liczenie odbędzie się z uwzględnioną nową wagą.

Wymagania

- Linux environment
- openvswitch service
- Python
- Java 8
- mininet
- ant

Implementacja

W folderze src znajdują się źródła jak i topologia, która zostaje wczytana przez symulator mininet. Aby uruchomić minineta z zewnętrznym kontrolerem należy wpisać:

```
sudo mn --custom=<topology_file.py> --topo=<topology> --controller=remote, ip=127.0.0.1, port=6653
```

Folder src to testy implementacji algorytmu Bellmana-Forda

Live Demo

