

IMSC HÁZI FELADAT

Szoftvertechnológia

Feladatvégrehajtás és dokumentáció

Tömöri Péter András

I4RZ00

2024. január 22.

TARTALOM

1. Saját Unit tesztek.....	2
2. Generált kódok	4
3. Tesztelés eredménye	10
4. Tesztek generálása.....	11
5. Generált tesztek eredménye	15
6. Hiba kijavítása.....	16

1. Saját Unit tesztek

Az egységtesztek készítésekor a megvalósítandó interfész mindkét függvényét külön tesztelem, így a bonyolultabb függvény problémáival tudjuk, hogy csak akkor foglalkozunk, ha már az élek hozzáadása megfelelően működik.

Így először az `addEdge(int, int, int)` függvényre néztem meg, milyen esetek lehetségesek:

1. normál lefutás: 10, 11, 0
2. hibás súly (felső korlát): `Exception 11, 12, 3`
3. hibás súly (alsó korlát): `Exception 12, 13, -2`
4. már létező él hozzáadása `Exception 13, 14, 1; 13, 14, 1`
5. helyes az előzőre: 14, 15, 0; 14, 15, 1; 15, 14, 1

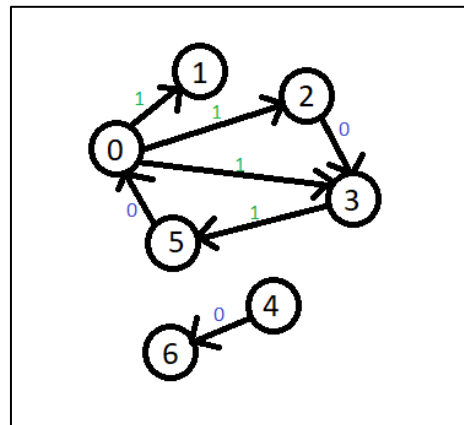
Az ötödik tesztet által azt feltételezem, hogy a „nem lehet két azonos él” megfogalmazásban az azonos csak arra vonatkozik, ha az él mindhárom paraméterében ugyanaz.

A `getShortestPathLength(int, int)` függvényt a következő gráfon teszteltem különböző esetekre:

Csúcsok: 0, 1, 2, 3, 4, 5, 6
1 súlyú élek: (0,1) (0,2) (0,3) (3,5)
0 súlyú élek: (5,0) (2,3), (4,6)

És a tesztek:

1. 0,4 kimenete -1
2. 4,6 kimenete 0
3. 6,4 kimenete -1
4. 5,2 kimenete 1
5. 2,1 kimenete 2
6. +3,0,0 éllel az 5. teszt: 1
7. 0,7 kimenete -1



A tesztek forráskódja:

```
import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

class PathfinderTest {
    private Pathfinder pathFinder;

    @BeforeEach
    void beforeEach() {
        pathFinder = new Graph();
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(0, 2, 1);
        pathFinder.addEdge(0, 3, 1);
        pathFinder.addEdge(3, 5, 1);
        pathFinder.addEdge(5, 0, 0);
        pathFinder.addEdge(2, 3, 0);
        pathFinder.addEdge(4, 6, 0);
    }
}
```

```

@Test
void singleEdgeTest() {
    pathFinder.addEdge(10, 11, 1);
    assertEquals(1, pathFinder.getShortestPathLength(10, 11));
}

@Test
void highWeightTest() {
    assertThrowsExactly(IllegalArgumentException.class, () ->
pathFinder.addEdge(11, 12, 3));
}

@Test
void lowWeightTest() {
    assertThrowsExactly(IllegalArgumentException.class, () ->
pathFinder.addEdge(12, 13, -2));
}

@Test
void sameEdgeTest() {
    pathFinder.addEdge(13, 14, 1);
    assertThrowsExactly(IllegalStateException.class, () ->
pathFinder.addEdge(13, 14, 1));
}

@Test
void sameNodesDifferentEdgesTest() {
    pathFinder.addEdge(14, 15, 0);
    pathFinder.addEdge(14, 15, 1);
    pathFinder.addEdge(15, 14, 1);
    assertEquals(1, pathFinder.getShortestPathLength(15, 14));
    assertEquals(0, pathFinder.getShortestPathLength(14, 15));
}

@Test
void noPathTest() {
    assertEquals(-1, pathFinder.getShortestPathLength(0, 4));
}

@Test
void singlePathTest() {
    assertEquals(0, pathFinder.getShortestPathLength(4, 6));
}

@Test
void wrongDirectionTest() {
    assertEquals(-1, pathFinder.getShortestPathLength(6, 4));
}

@Test
void twoEdgeTest() {
    assertEquals(1, pathFinder.getShortestPathLength(5, 2));
}

@Test
void longRouteTest() {
    assertEquals(2, pathFinder.getShortestPathLength(2, 1));
}

@Test

```

```

void moreOptionTest() {
    pathFinder.addEdge(3, 0, 0);
    assertEquals(1, pathFinder.getShortestPathLength(2, 1));
}

@Test
void noNodeTest() {
    assertEquals(-1, pathFinder.getShortestPathLength(0, 7));
}
}

```

2. Generált kódok

A gódkeneráláshoz a ChatGPT-t, a Bing AI-t és a Google Bard-ot használtam.

Mindháromnak az alábbi promptokból álló üzenetet küldtem el:

Implement an interface called PathFinder in Java. The interface finds the shortest path in a weighted directed graph.

The function addEdge(int from, int to, int weight) adds an edge to the directed graph. Parameter from is the ID of the node at the start of the edge. Parameter to is the ID of the node at the end of the edge. Parameter weight is the weight of the edge, which can only be 1 or 0. If the edge weight is invalid, the function throws IllegalArgumentException. If the edge was already present in the graph the function throws IllegalStateException.

The other function of Pathfinder is getShortestPathLength(int source, int target). This function calculates the weight of the shortest path in the graph between the two nodes. If this shortest path exists it returns the value of it, but if there is no path from the source to the target it returns -1.

ChatGPT válasza:

```

import java.util.*;

interface PathFinder {
    void addEdge(int from, int to, int weight) throws
    IllegalArgumentException, IllegalStateException;
    int getShortestPathLength(int source, int target);
}

class Graph implements PathFinder {
    private Map<Integer, Map<Integer, Integer>> graph;

    public Graph() {
        graph = new HashMap<>();
    }
}

```

```

        @Override
        public void addEdge(int from, int to, int weight) throws
        IllegalArgumentException, IllegalStateException {
            if (weight != 0 && weight != 1) {
                throw new IllegalArgumentException("Invalid edge weight. It
must be 0 or 1.");
            }

            graph.computeIfAbsent(from, k -> new HashMap<>());

            if (graph.get(from).containsKey(to)) {
                throw new IllegalStateException("Edge already exists in the
graph.");
            }

            graph.get(from).put(to, weight);
        }

        @Override
        public int getShortestPathLength(int source, int target) {
            if (!graph.containsKey(source) || !graph.containsKey(target)) {
                return -1; // No path if either source or target is not in
the graph
            }

            Map<Integer, Integer> distance = new HashMap<>();
            PriorityQueue<int[]> minHeap = new
PriorityQueue<>(Comparator.comparingInt(a -> a[1]));

            minHeap.offer(new int[]{source, 0});
            distance.put(source, 0);

            while (!minHeap.isEmpty()) {
                int[] current = minHeap.poll();
                int currentNode = current[0];
                int currentDistance = current[1];

                if (currentNode == target) {
                    return currentDistance; // Found shortest path to target
                }

                if (graph.containsKey(currentNode)) {
                    for (Map.Entry<Integer, Integer> neighbor :
graph.get(currentNode).entrySet()) {
                        int nextNode = neighbor.getKey();
                        int weight = neighbor.getValue();
                        int newDistance = currentDistance + weight;

                        if (!distance.containsKey(nextNode) || newDistance <
distance.get(nextNode)) {
                            distance.put(nextNode, newDistance);
                            minHeap.offer(new int[]{nextNode, newDistance});
                        }
                    }
                }
            }

            return -1; // No path found from source to target
        }
    }
}

```

Bing AI válasza:

```
import java.util.*;

interface Pathfinder {
    void addEdge(int from, int to, int weight) throws
        IllegalArgumentException, IllegalStateException;
    int getShortestPathLength(int source, int target);
}

class Graph implements Pathfinder {
    private Map<Integer, List<int[]>> graph = new HashMap<>();

    public void addEdge(int from, int to, int weight) throws
        IllegalArgumentException, IllegalStateException {
        if (weight != 0 && weight != 1) {
            throw new IllegalArgumentException("Invalid edge weight");
        }
        graph.putIfAbsent(from, new ArrayList<>());
        List<int[]> edges = graph.get(from);
        for (int[] edge : edges) {
            if (edge[0] == to) {
                throw new IllegalStateException("Edge already exists");
            }
        }
        edges.add(new int[]{to, weight});
    }

    public int getShortestPathLength(int source, int target) {
        Map<Integer, Integer> distances = new HashMap<>();
        PriorityQueue<int[]> pq = new
        PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
        pq.offer(new int[]{source, 0});
        while (!pq.isEmpty()) {
            int[] curr = pq.poll();
            int node = curr[0], dist = curr[1];
            if (distances.containsKey(node)) {
                continue;
            }
            distances.put(node, dist);
            if (node == target) {
                return dist;
            }
            for (int[] edge : graph.getOrDefault(node,
                Collections.emptyList())) {
                int neighbor = edge[0], weight = edge[1];
                if (!distances.containsKey(neighbor)) {
                    pq.offer(new int[]{neighbor, dist + weight});
                }
            }
        }
        return -1;
    }
}
```

Google Bard válasza:

```
import java.util.*;

public class DijkstraPathFinder implements Pathfinder {
```

```

        private int[][] adjacencyMatrix; // Adjacency matrix representing the
graph
        private int[] distance; // Distance to each node from the source
        private boolean[] visited; // Visited nodes
        private PriorityQueue<Node> pq; // Priority queue for maintaining the
unvisited nodes

        public DijkstraPathFinder(int numNodes) {
            adjacencyMatrix = new int[numNodes][numNodes];
            distance = new int[numNodes];
            visited = new boolean[numNodes];
            pq = new PriorityQueue<>(Comparator.comparingInt(node ->
node.distance));
        }

        @Override
        public void addEdge(int from, int to, int weight) throws
IllegalArgumentException, IllegalStateException {
            if (weight != 1 && weight != 0) {
                throw new IllegalArgumentException("Edge weight must be 1 or
0");
            }
            if (adjacencyMatrix[from][to] != 0) {
                throw new IllegalStateException("Edge already exists");
            }
            adjacencyMatrix[from][to] = weight;
        }

        @Override
        public int getShortestPathLength(int source, int target) {
            if (source < 0 || source >= adjacencyMatrix.length || target < 0
|| target >= adjacencyMatrix.length) {
                throw new IllegalArgumentException("Invalid node IDs");
            }

            for (int i = 0; i < adjacencyMatrix.length; i++) {
                distance[i] = Integer.MAX_VALUE; // Initialize distances to
infinity
            }
            distance[source] = 0; // Distance to the source is 0

            pq.add(new Node(source, 0)); // Add the source to the priority
queue

            while (!pq.isEmpty()) {
                Node curr = pq.poll(); // Get the node with the minimum
distance
                visited[curr.node] = true; // Mark the node as visited

                for (int i = 0; i < adjacencyMatrix.length; i++) {
                    if (adjacencyMatrix[curr.node][i] != 0 && !visited[i]) {
                        int newDistance = distance[curr.node] +
adjacencyMatrix[curr.node][i];
                        if (newDistance < distance[i]) {
                            distance[i] = newDistance; // Update the distance
if shorter path found
                            pq.add(new Node(i, newDistance)); // Add the
updated node to the priority queue
                        }
                    }
                }
            }
        }
    }
}

```

```

    }

    return visited[target] ? distance[target] : -1; // Return the
shortest path length or -1 if no path exists

```

Programkódok értékelése:

A ChatGPT le is írta, hogy a Dijkstra algoritmussal valósítja meg. A gráfot pedig HashMappel tárolja, és azt szépen kezeli, valamint a kivételeket sem hagyta figyelmen kívül, látszólag kitért minden esetre. Továbbá az Override annotációkat ki kellett vennem.

A Bing AI gráftárolása egy kicsit eltérő, de a kód hasonlóan néz ki. Habár nekem ez kicsit nehezebben volt átlátható, egy minimális kommentezés ráférne.

A Google Bard elsőre csak a fejléceket készítette el, de szépen kellett kérni és megírta a kódot is, elmagyarázva, hogy a Dijkstra algoritmust használja benne szomszédossági mátrix-szal.

A kód szépen kommentelt, viszont komoly hiba, hogy a második függvényben is dob IllegalArgumentException-t, de azt már sehol sem kezeli. Illetve a PriorityQueue-ben Node-okat kezel, amit nem definiált sehol.

A problémákat közöltem vele:

I have some issues with the code, that you should correct. The pq PriorityQueue contains "Node" type, but Node is declared nowhere. You should implement it or substitute with something else that also works. The other problem is that your implementation of getShortestPathLength throws Exception that is handled nowhere.

És javította a kódot az alábbi szerint:

```

import java.util.*;

class Node {

    private int node; // Node ID
    private int distance; // Distance from source node

    public Node(int node, int distance) {
        this.node = node;
        this.distance = distance;
    }

    public int getNode() {
        return node;
    }

    public int getDistance() {
        return distance;
    }
}

public class DijkstraPathFinder implements Pathfinder {

```



```

        private int[][] adjacencyMatrix; // Adjacency matrix representing the
graph
        private int[] distance; // Distance to each node from the source
        private boolean[] visited; // Visited nodes
        private PriorityQueue<Node> pq; // Priority queue for maintaining the
unvisited nodes

        public DijkstraPathFinder(int numNodes) {
            adjacencyMatrix = new int[numNodes][numNodes];
            distance = new int[numNodes];
            visited = new boolean[numNodes];
            pq = new PriorityQueue<>(Comparator.comparingInt(node ->
node.distance));
        }

        @Override
        public void addEdge(int from, int to, int weight) throws
IllegalArgumentException, IllegalStateException {
            if (weight != 1 && weight != 0) {
                throw new IllegalArgumentException("Edge weight must be 1 or
0");
            }
            if (adjacencyMatrix[from][to] != 0) {
                throw new IllegalStateException("Edge already exists");
            }
            adjacencyMatrix[from][to] = weight;
        }

        @Override
        public int getShortestPathLength(int source, int target) throws
IllegalArgumentException {
            if (source < 0 || source >= adjacencyMatrix.length || target < 0
|| target >= adjacencyMatrix.length) {
                throw new IllegalArgumentException("Invalid node IDs");
            }

            for (int i = 0; i < adjacencyMatrix.length; i++) {
                distance[i] = Integer.MAX_VALUE; // Initialize distances to
infinity
            }
            distance[source] = 0; // Distance to the source is 0

            pq.add(new Node(source, 0)); // Add the source to the priority
queue

            while (!pq.isEmpty()) {
                Node curr = pq.poll(); // Get the node with the minimum
distance
                visited[curr.getNode()] = true; // Mark the node as visited

                for (int i = 0; i < adjacencyMatrix.length; i++) {
                    if (adjacencyMatrix[curr.getNode()][i] != 0
&& !visited[i]) {
                        int newDistance = distance[curr.getNode()] +
adjacencyMatrix[curr.getNode()][i];
                        if (newDistance < distance[i]) {
                            distance[i] = newDistance; // Update the distance
if shorter path found
                            pq.add(new Node(i, newDistance)); // Add the
updated node to the priority queue

```

```

    }
    }
}

if (visited[target]) {
    return distance[target]; // Return the shortest path length
} else {
    throw new RuntimeException("No path exists from source to
target"); // Handle the exception properly

```

Mivel ez a kód a végén kivételt dob a -1 helyett, ezért ismét jeleztem, hogy valami nem stimmel és kijavította:

```

...
return visited[target] ? distance[target] : -1; // Return the shortest
path length or -1 if no path exists

```

Ezen kívül a node.distance láthatósága miatt át kellett írnom néhány helyen, hogy használja a gettereket. Illetve innen is kiszedtem az Override-okat.

3. Tesztelés eredménye

	helyes/összes	döntési cov.	utasítás cov.
ChatGPT	7/12	1 nem volt	100%
Bing AI	11/12	1 nem volt	99,4%
Google Bard	7/12	2 nem volt	97.5%

Google Bardnak meg kellett a konstruktorába adni, hogy hány csúcsa van, és a csúcsok azonosítójának is bele kellett esnie ebbe a tartományba. Gyenge implementációs megoldás.

A Bing AI teljesített a legjobban a teszteknel, csak a sameNodesDifferentEdgesTest()-en bukott el, azért, mert ő már akkor is egyformának tekint két élet, ha a két csúcs megegyezik (sorrendtől és súlytól függetlenül). A döntési lefedettségekből arra következtetek, hogy még lehetett volna egy tesztesetet írni valamilyen funkcióra, de az is lehet, hogy hibás implementáció miatt nem került sor valamire sosem (például a Google Bard második függvényében Exception dobására, ami csak hibás függvényhívás esetén történhetne).

4. Tesztek generálása

A tesztkészítéshez bemásoltam az interfészt a kommentekkel és megkértem, hogy készítsenek JUnit 5 teszteket, amik 100%-os kódlefedettséget biztosítanak.

ChatGPT válasza:

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class PathfinderTest {

    private Pathfinder pathFinder;

    //Szép megoldás a Before
    @Before
    public void setUp() {
        // Initialize your Pathfinder implementation here
        // pathFinder = new YourPathFinderImplementation();
    //kiegészítettem az inicializálással:
        pathFinder=new Graph();
    }

    //Él helyes hozzáadását teszelne csak nem tudja hogyan
    @Test
    public void testAddEdge() {
        // Test adding a valid edge
        pathFinder.addEdge(1, 2, 1);
        assertTrue(true); // Change this assertion based on your
implementation

        // Test adding an edge with weight 0
        pathFinder.addEdge(2, 3, 0);
        assertTrue(true); // Change this assertion based on your
implementation

        // Add more test cases for edge addition
    }

    //Nem sikerült JUnit5-ben de nem nagy baj
    //Helyesen teszteli a kivételes eseteket
    @Test(expected = IllegalArgumentException.class)
    public void testAddEdgeInvalidWeight() {
        // Test adding an edge with invalid weight
        pathFinder.addEdge(1, 2, 2);
    }

    @Test(expected = IllegalStateException.class)
    public void testAddEdgeDuplicate() {
        // Test adding a duplicate edge
        pathFinder.addEdge(1, 2, 1);
        pathFinder.addEdge(1, 2, 1);
    }

    //A program alap működésére jó, de még kéne nézni olyat is amikor több
    útból kell ténylegesen a rövidebbet választania
```

```

@Test
public void testGetShortestPathLength() {
    // Test getting the shortest path length
    pathFinder.addEdge(1, 2, 1);
    pathFinder.addEdge(2, 3, 1);
    pathFinder.addEdge(3, 4, 1);
    int shortestPathLength = pathFinder.getShortestPathLength(1, 4);
    assertEquals(3, shortestPathLength); // Change this assertion
    based on your implementation

    // Test when there is no path
    pathFinder.addEdge(5, 6, 1);
    shortestPathLength = pathFinder.getShortestPathLength(1, 6);
    assertEquals(-1, shortestPathLength); // Change this assertion
    based on your implementation

    // Add more test cases for different scenarios
    //Fontosabb hiány: a tényleges Dijkstra algo. kihívás elé tétele
    //Kéne néhány helyre 0 hosszú élt is rakni
    //Illetve az irányítottság vizsgálatára
}
}

```

Bing AI válasza:

To achieve 100% code coverage for an interface's implementation, you need to write unit tests for the classes that implement the interface. Since an interface does not do anything on its own, there is no reason and no way to test it 12.

You can write unit tests for the individual implementations of the interface like you normally would. To enforce an interface's behavior generally, you can write a helper class or method that checks the invariants of an arbitrary implementation, then have a test in the implementation's unit tests that calls that helper 1.

I hope this helps!

Újabb kérésre 2 teszt, majd sikerült többet:

```

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

//Élhozzáadás szép tesztelése kivételekkel együtt
//Célszerűbbnek tartanám szétszedni 2/3 esetre
public class PathFinderTest {
    @Test
    public void testAddEdge() {
        PathFinder pathFinder = new PathFinderImpl();
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(1, 2, 0);
        pathFinder.addEdge(2, 3, 1);
        Assertions.assertThrows(IllegalArgumentException.class, () ->
            pathFinder.addEdge(3, 4, 2));
        Assertions.assertThrows(IllegalStateException.class, () ->
            pathFinder.addEdge(2, 3, 1));
    }

    //Jó teszt a szimpla útra és az irányítottságra is
    @Test
    public void testGetShortestPathLength() {
        PathFinder pathFinder = new PathFinderImpl();
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(1, 2, 0);
    }
}

```

```

        pathFinder.addEdge(2, 3, 1);
        Assertions.assertEquals(2, pathFinder.getShortestPathLength(0,
3));
        Assertions.assertEquals(-1, pathFinder.getShortestPathLength(3,
0));
    }
    //Csinált végül külön is, de tulajdonképpen ez az első ismétlése
    //Úgy tűnik, amikor azt mondtam neki, hogy írjon még 3mat, hogy jobb
    legyen a lefedettség, az volt a fő cél előtte, hogy legyen 3
    @Test
    public void testAddEdgeInvalidWeight() {
        PathFinder pathFinder = new PathFinderImpl();
        Assertions.assertThrows(IllegalArgumentException.class, () ->
pathFinder.addEdge(0, 1, 2));
    }

    @Test
    public void testAddEdgeAlreadyPresent() {
        PathFinder pathFinder = new PathFinderImpl();
        pathFinder.addEdge(0, 1, 1);
        Assertions.assertThrows(IllegalStateException.class, () ->
pathFinder.addEdge(0, 1, 1));
    }

    //Szintén csak ismétlés
    @Test
    public void testGetShortestPathLengthNoPath() {
        PathFinder pathFinder = new PathFinderImpl();
        pathFinder.addEdge(0, 1, 1);
        pathFinder.addEdge(1, 2, 0);
        pathFinder.addEdge(2, 3, 1);
        Assertions.assertEquals(-1, pathFinder.getShortestPathLength(3,
1));
    }
}

```

Google Bard válasza:

```

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

/**
 * Unit tests for the `PathFinder` interface.
 */
class PathFinderTests {

    /**
     * Test adding a valid edge.
     */
    @Test
    public void testAddValidEdge() {
        PathFinder pathfinder = new PathFinderImpl();
        pathfinder.addEdge(0, 1, 1);
        //Olyan függvénnyel akart tesztelni, amit nem implementált, ezért ezt
        kikommenteltem
        //Assertions.assertTrue(pathfinder.hasEdge(0, 1));
    }

    /**
     * Test adding an invalid edge with an invalid weight.
     */
}

```

```

    */
    @Test
    public void testAddInvalidEdgeWithInvalidWeight() {
        Pathfinder pathfinder = new PathfinderImpl();
        Assertions.assertThrows(IllegalArgumentException.class, () ->
            pathfinder.addEdge(0, 1, -1));
    }

    /**
     * Test adding an invalid edge that already exists.
     */
    @Test
    public void testAddInvalidEdgeThatAlreadyExists() {
        Pathfinder pathfinder = new PathfinderImpl();
        pathfinder.addEdge(0, 1, 1);
        Assertions.assertThrows(IllegalStateException.class, () ->
            pathfinder.addEdge(0, 1, 2));
    }
    //Kivételes eseteket szépen kezel, bár az nem a legjobb, hogy az
    //ismétlődő csúcs tesztjénél invalid súlyt is ad meg, gondot okozhat.
    //(Ráadásul az én értelmezésem szerint ennek amúgy sem kéne megegyező éleknek
    //lenni)

    /**
     * Test calculating the shortest path length for a simple graph with
     * a single path.
     */
    @Test
    public void testCalculateShortestPathLengthForSimpleGraph() {
        Pathfinder pathfinder = new PathfinderImpl();
        //A saját tesztjében érvénytelen függvényhívást végez :/
        pathfinder.addEdge(0, 1, 2);
        pathfinder.addEdge(1, 2, 1);

        int shortestPathLength = pathfinder.getShortestPathLength(0, 2);
        //Így nehezen jutunk el idáig
        Assertions.assertEquals(3, shortestPathLength);
    }

    /**
     * Test calculating the shortest path length for a graph with
     * multiple paths and a negative weight.
     */
    @Test
    public void
    testCalculateShortestPathLengthForGraphWithMultiplePathsAndNegativeWeight
    () {
        Pathfinder pathfinder = new PathfinderImpl();
        pathfinder.addEdge(0, 2, 2);
        pathfinder.addEdge(0, 1, 1);
        //Ismét érvénytelen él, úgy tűnik az interface headerből kevésbé érti meg
        //a feladatot, mintha rövid mondatokba szedem ugyanazt.
        pathfinder.addEdge(1, 2, -4);

        int shortestPathLength = pathfinder.getShortestPathLength(0, 2);
        Assertions.assertEquals(-4, shortestPathLength);
    }

    /**
     * Test calculating the shortest path length for a graph with a cycle
     * and a negative weight.

```

```

        */
        @Test
        public void
testCalculateShortestPathLengthForGraphWithCycleAndNegativeWeight() {
    Pathfinder pathfinder = new PathfinderImpl();
    pathfinder.addEdge(0, 1, 2);
    pathfinder.addEdge(1, 2, -2);
    pathfinder.addEdge(2, 0, -1);

    int shortestPathLength = pathfinder.getShortestPathLength(0, 2);
    Assertions.assertEquals(-1, shortestPathLength);
}

/**
 * Test calculating the shortest path length for a disconnected
graph.
 */
@Test
public void testCalculateShortestPathLengthForDisconnectedGraph() {
    Pathfinder pathfinder = new PathfinderImpl();
    pathfinder.addEdge(0, 1, 2);

    int shortestPathLength = pathfinder.getShortestPathLength(1, 2);
    Assertions.assertEquals(-1, shortestPathLength);
}
}

```

5. Generált tesztek eredménye

	helyes/összes	döntési cov.	utasítás cov.
ChatGPT	3/4	5/16	36%
Bing AI	5/5	14/16	97,1%
Google Bard	2/7	2/16	23,8%

Vélemények: Habár a Bing AI-t volt a legnehezebb meggyőzni arról, hogy implementáció nélkül is van értelme tesztet készíteni, és utána is tulajdonképpen csak lemásolta a tesztjeit, hogy kétszer annyi legyen, mégis az ő tesztkészlete volt a legalkalmasabb arra, hogy ellenőrizze a fő lefutásokat (az egyébként jó implementációján). Szerencséjére nem készített olyan tesztet, amiben azt ellenőrizné, amit félreértelmezett a kódkészítéskor. Érdekességből lefuttattam a ChatGPT és a Google Bard tesztjeit a Bing AI implementációján: a Google Bard-dal 31%os utasítás lefedettség és 2/7 teszt, míg a ChatGPT-vel 4/4 teszt és 97.1%-os utasítás lefedettség lett.

Ebből arra merek következtetni, hogy habár a ChatGPT az implementációt valahol erősen elrontotta, teszteket mégis kielégítően tudott hozzá írni. Ezzel szemben a Bard-ot nem javasolnám senkinek szoftverfejlesztési eszköznek.

A tesztekben saját kommentjeimmel jegyeztem fel, melyik mi célt szolgálhat, hol lehet esetleg a tesztben hiba (sajnos egész sok ilyen is akadt).

A tesztek végig tekintése után a saját tesztkészletemmel összehasonlítva: Sok apróságra is kitértem, amit alapvetően az implementáció során nehéz elrontani. Ugyanakkor van 2-3 fontosabb teszt, amik szerintem nem nélkülözhetőek, és (szinte) csak az enyémekben szerepeltek.

6. Hiba kijavítása

A Bing AI egészen jól teljesítette a feladatokat, így szerintem az ő implementációját a legérdekesebb kijavíttatni, főleg mivel csak 1 hibát vétett: Ha két él végpontja megegyezik (sorrendtől és élsúlytól függetlenül), az ő programja arra is kivételt dob.

Beszélgetés:

In my interpretation, two edges in a directed graph are only the same, if the two nodes, the edges' direction and weight are all the same.
Correct your implementation in this way. Therefore `IllegalStateException` should occur less often and you might also change other parts of the code.

- ✓ how to implement an interface called `PathFinder` in Java with edge comparison keresése
- ✓ Válaszok létrehozása...

Thank you for your feedback. Here is an updated implementation of the `PathFinder` interface in Java that takes into account your interpretation:

Java

```
import java.util.*;

interface PathFinder {
    void addEdge(int from, int to, int weight) throws
        IllegalArgumentException, IllegalStateException;
    int getShortestPathLength(int source, int target);
}
```

```

class Graph implements Pathfinder {
    private Map<Integer, Set<int[]>> graph = new HashMap<>();

    public void addEdge(int from, int to, int weight) throws
    IllegalArgumentException, IllegalStateException {
        if (weight != 0 && weight != 1) {
            throw new IllegalArgumentException("Invalid edge
weight");
        }
        graph.putIfAbsent(from, new HashSet<>());
        Set<int[]> edges = graph.get(from);
        for (int[] edge : edges) {
            if (edge[0] == to && edge[1] == weight) {
                throw new IllegalStateException("Edge already
exists");
            }
        }
        edges.add(new int[]{to, weight});
    }

    public int getShortestPathLength(int source, int target) {
        Map<Integer, Integer> distances = new HashMap<>();
        PriorityQueue<int[]> pq = new PriorityQueue<>
(Comparator.comparingInt(a -> a[1]));
        pq.offer(new int[]{source, 0});
        while (!pq.isEmpty()) {

```

```

        int[] curr = pq.poll();
        int node = curr[0], dist = curr[1];
        if (distances.containsKey(node)) {
            continue;
        }
        distances.put(node, dist);
        if (node == target) {
            return dist;
        }
        for (int[] edge : graph.getOrDefault(node,
Collections.emptySet())) {
            int neighbor = edge[0], weight = edge[1];
            if (!distances.containsKey(neighbor)) {
                pq.offer(new int[]{neighbor, dist + weight});
            }
        }
    }
    return -1;
}
}

```

Mesterséges intelligencia által létrehozott kód. Ellenőrizze és körültekintően használja. [További információ a GYIK-ről.](#)

The `Graph` class implements the `PathFinder` interface and provides implementations for the `addEdge` and `getShortestPathLength` methods. The `addEdge` method adds an edge to the graph with the specified start and end nodes and weight. If the weight is not 0 or 1, an `IllegalArgumentException` is thrown. If the edge already exists, an `IllegalStateException` is thrown. The `getShortestPathLength` method calculates the shortest path between the specified source and target nodes using Dijkstra's algorithm and returns the length of the path. If there is no path from the source to the target, it returns -1.

I hope this updated implementation meets your requirements. Let me know if you have any further questions.

További információ



1



baeldung.com

2



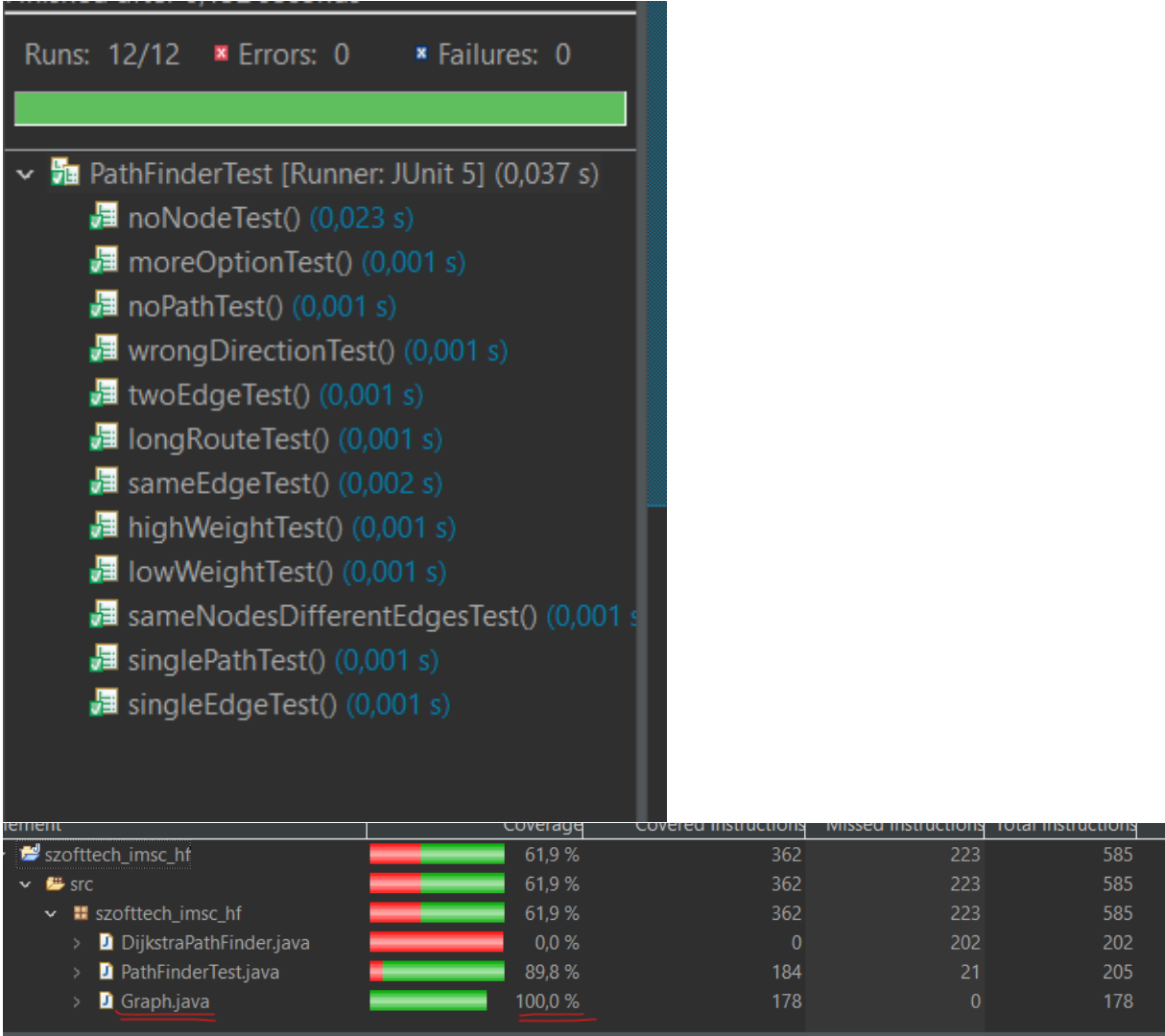
geeksforgeeks.org



2 / 30 •

Tekintve, hogy a Bing rögtön megjavította a hibát, a beszélgetés nem tartott sok üzenetváltáson keresztül.

Az új kód eredménye a saját tesztjeimmel:



```

112 class Graph implements PathFinder {
113     private Map<Integer, Set<int[]>> graph = new HashMap<>();
114
115     public void addEdge(int from, int to, int weight) throws IllegalArgumentException, IllegalStateException {
116         if (weight != 0 && weight != 1) {
117             throw new IllegalArgumentException("Invalid edge weight");
118         }
119         graph.putIfAbsent(from, new HashSet<>());
120         Set<int[]> edges = graph.get(from);
121         for (int[] edge : edges) {
122             if (edge[0] == to && edge[1] == weight) {
123                 throw new IllegalStateException("Edge already exists");
124             }
125         }
126         edges.add(new int[]{to, weight});
127     }
128
129     public int getShortestPathLength(int source, int target) {
130         Map<Integer, Integer> distances = new HashMap<>();
131         PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
132         pq.offer(new int[]{source, 0});
133         while (!pq.isEmpty()) {
134             int[] curr = pq.poll();
135             int node = curr[0], dist = curr[1];
136             if (distances.containsKey(node)) {
137                 continue;
138             }
139             distances.put(node, dist);
140             if (node == target) {
141                 return dist;
142             }
143             for (int[] edge : graph.getOrDefault(node, Collections.emptySet())) {
144                 int neighbor = edge[0], weight = edge[1];
145                 if (!distances.containsKey(neighbor)) {
146                     pq.offer(new int[]{neighbor, dist + weight});
147                 }
148             }
149         }
150         return -1;

```

Tehát minden 100%-os lett :)