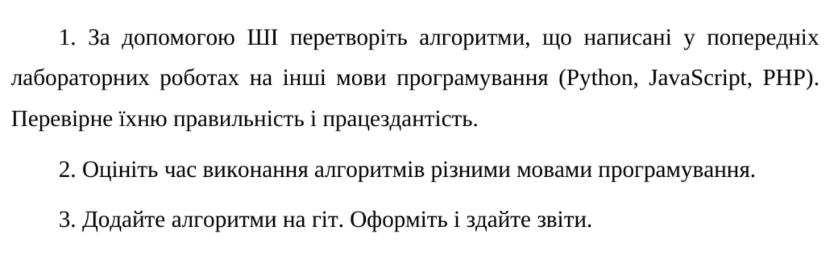
**ЗВІТ**

про виконання лабораторної роботи 8

«**Реалізація шкільних олімпіадних задач різними мовами програмування**»  
з дисципліни  
«Методика розв'язування олімпіадних задач» студента(ки) групи ІН-2327Б  
Чухрая Олександра Васильовича

**Умова завдання:**

****

**Виконання:**

JavaScript:

class Edge {

constructor(to, weight) {

this.to = to;

this.weight = weight;

}

}

class Graph {

constructor() {

this.adjacencyList = {};

}

addEdge(from, to, weight) {

if (!this.adjacencyList[from]) this.adjacencyList[from] = [];

if (!this.adjacencyList[to]) this.adjacencyList[to] = [];

this.adjacencyList[from].push(new Edge(to, weight));

this.adjacencyList[to].push(new Edge(from, weight)); // Undirected

}

dijkstra(start) {

const distances = {};

const visited = new Set();

const pq = new Set();

for (const node in this.adjacencyList) {

distances[node] = Infinity;

}

distances[start] = 0;

pq.add([0, start]);

while (pq.size > 0) {

const minNode = Array.from(pq).reduce((min, curr) => (curr[0] < min[0] ? curr : min));

pq.delete(minNode);

const [currentDistance, currentNode] = minNode;

if (visited.has(currentNode)) continue;

visited.add(currentNode);

for (const edge of this.adjacencyList[currentNode]) {

const newDist = currentDistance + edge.weight;

if (newDist < distances[edge.to]) {

distances[edge.to] = newDist;

pq.add([newDist, edge.to]);

}

}

}

console.log(`Shortest distances from node ${start}:`);

for (const node in distances) {

console.log(`To ${node} -> ${distances[node]}`);

}

}

}

const graph = new Graph();

graph.addEdge(3, 1, 3);

graph.addEdge(3, 6, 4);

graph.addEdge(3, 5, 5);

graph.addEdge(6, 1, 1);

graph.addEdge(6, 2, 11);

graph.addEdge(6, 4, 2);

graph.addEdge(6, 5, 6);

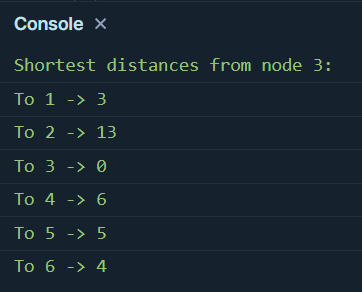
graph.addEdge(5, 4, 5);

graph.addEdge(4, 2, 8);

graph.addEdge(1, 2, 10);

graph.dijkstra(3);

Результат:



PHP:

<?php

class Edge {

public $to, $weight;

public function \_\_construct($to, $weight) {

$this->to = $to;

$this->weight = $weight;

}

}

class Graph {

private $adjacencyList = [];

public function addEdge($from, $to, $weight) {

if (!isset($this->adjacencyList[$from])) $this->adjacencyList[$from] = [];

if (!isset($this->adjacencyList[$to])) $this->adjacencyList[$to] = [];

$this->adjacencyList[$from][] = new Edge($to, $weight);

$this->adjacencyList[$to][] = new Edge($from, $weight); // Undirected

}

public function dijkstra($start) {

$distances = [];

$visited = [];

$pq = [];

foreach ($this->adjacencyList as $node => $\_) {

$distances[$node] = INF;

}

$distances[$start] = 0;

$pq[] = [$start, 0];

while (!empty($pq)) {

usort($pq, fn($a, $b) => $a[1] <=> $b[1]);

[$currentNode, $currentDistance] = array\_shift($pq);

if (isset($visited[$currentNode])) continue;

$visited[$currentNode] = true;

foreach ($this->adjacencyList[$currentNode] as $edge) {

$newDist = $currentDistance + $edge->weight;

if ($newDist < $distances[$edge->to]) {

$distances[$edge->to] = $newDist;

$pq[] = [$edge->to, $newDist];

}

}

}

echo "Shortest distances from node $start:\n";

foreach ($distances as $node => $dist) {

echo "To $node -> $dist\n";

}

}

}

$graph = new Graph();

$graph->addEdge(3, 1, 3);

$graph->addEdge(3, 6, 4);

$graph->addEdge(3, 5, 5);

$graph->addEdge(6, 1, 1);

$graph->addEdge(6, 2, 11);

$graph->addEdge(6, 4, 2);

$graph->addEdge(6, 5, 6);

$graph->addEdge(5, 4, 5);

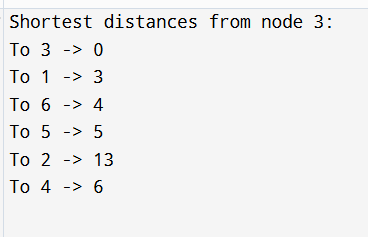
$graph->addEdge(4, 2, 8);

$graph->addEdge(1, 2, 10);

$graph->dijkstra(3);

?>

Результат:



Python:

import heapq

from collections import defaultdict

class Edge:

def \_\_init\_\_(self, to, weight):

self.to = to

self.weight = weight

class Graph:

def \_\_init\_\_(self):

self.adjacency\_list = defaultdict(list)

def add\_edge(self, from\_node, to\_node, weight):

self.adjacency\_list[from\_node].append(Edge(to\_node, weight))

self.adjacency\_list[to\_node].append(Edge(from\_node, weight)) # Undirected

def dijkstra(self, start):

distances = {node: float('inf') for node in self.adjacency\_list}

distances[start] = 0

visited = set()

pq = [(0, start)]

while pq:

current\_distance, current\_node = heapq.heappop(pq)

if current\_node in visited:

continue

visited.add(current\_node)

for edge in self.adjacency\_list[current\_node]:

new\_dist = current\_distance + edge.weight

if new\_dist < distances[edge.to]:

distances[edge.to] = new\_dist

heapq.heappush(pq, (new\_dist, edge.to))

print(f"Shortest distances from node {start}:")

for node, dist in distances.items():

print(f"To {node} -> {dist}")

graph = Graph()

graph.add\_edge(3, 1, 3)

graph.add\_edge(3, 6, 4)

graph.add\_edge(3, 5, 5)

graph.add\_edge(6, 1, 1)

graph.add\_edge(6, 2, 11)

graph.add\_edge(6, 4, 2)

graph.add\_edge(6, 5, 6)

graph.add\_edge(5, 4, 5)

graph.add\_edge(4, 2, 8)

graph.add\_edge(1, 2, 10)

graph.dijkstra(3)

Результат:

