**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**Лабораторна робота №7**

**Алгоритми на Графах. Алгоритм Дейкстри**

Виконав студент 1 курсу

спеціальності “Інженерія

програмного забезпечення”

Дєлов В’ячеслав Едуардович

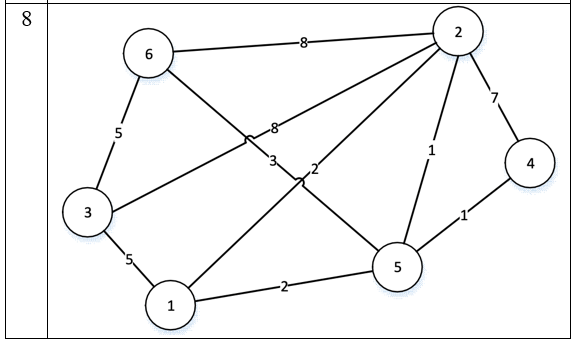
**Ужгород-2025**

**Мета:** навчитися реалізовувати алгоритм Дейкстри для пошуку найкоротшого шляху від однієї вершини графа до інших

**Завдання до практичної роботи**

1. Написати код програми, яка реалізує алгоритм Дейкстри, відповідно до обраного варіанту. Варіант, де вершини виділені повинен виконувати конкретне завдання. Там, де вершини не виділені повинна бути можливість задати вершин для пошуку шляху.
2. Протестувати роботу програми використовуючи різні вхідні дані.
3. Обґрунтувати хід виконання програми.
4. Оформити звіт та завантажити його в системі електронного навчання ДВНЗ «УжНУ» в установлений термін.
5. Підготувати відповіді на контрольні питання.

**Варіант 18 (8)**

****

**Хід роботи:**

Код програми:

class Node {

    constructor(id) {

      this.id = id;

      this.edges = [];

      this.parents = new Map(); // сусідній вузол -> ребро

    }

}

class Edge {

    constructor(adjacentNode, weight) {

      this.adjacentNode = adjacentNode;

      this.weight = weight;

    }

}

function initHashTables(start, graph, unprocessedNodes, timeToNodes) {

    for (let node of graph.values()) {

      unprocessedNodes.add(node);

      timeToNodes.set(node, Infinity);

    }

    timeToNodes.set(start, 0);

}

  function getNodeWithMinTime(unprocessedNodes, timeToNodes) {

    let minNode = null;

    let minTime = Infinity;

    for (let node of unprocessedNodes) {

      const time = timeToNodes.get(node);

      if (time < minTime) {

        minTime = time;

        minNode = node;

      }

    }

    return minNode;

}

function calculateTimeToEachNode(unprocessedNodes, timeToNodes) {

    while (unprocessedNodes.size > 0) {

      const node = getNodeWithMinTime(unprocessedNodes, timeToNodes);

      if (timeToNodes.get(node) === Infinity) return;

      for (let edge of node.edges) {

        const adjacentNode = edge.adjacentNode;

        if (unprocessedNodes.has(adjacentNode)) {

          const timeToCheck = timeToNodes.get(node) + edge.weight;

          if (timeToCheck < timeToNodes.get(adjacentNode)) {

            timeToNodes.set(adjacentNode, timeToCheck);

            adjacentNode.parents.set(node, edge);

          }

        }

      }

      unprocessedNodes.delete(node);

    }

}

function getShortestPath(start, end, timeToNodes) {

    const path = [];

    let current = end;

    while (current !== start) {

      path.unshift(current); //на початок

      // відбір батька з найменшим часом

      let next = null;

      for (let [parent, edge] of current.parents.entries()) {

        if (timeToNodes.has(parent) &&

            timeToNodes.get(parent) + edge.weight === timeToNodes.get(current)) {

          next = parent;

          break;

        }

      }

      if (!next) {

        console.log("Шлях не знайдено!");

        return null;

      }

      current = next;

    }

    //стартовий вузол на початок

    path.unshift(start);

    return path;

}

// Main

function getShortestPathMain(graph, start, end) {

    const unprocessedNodes = new Set();

    const timeToNodes = new Map();

    initHashTables(start, graph, unprocessedNodes, timeToNodes);

    calculateTimeToEachNode(unprocessedNodes, timeToNodes);

    if (timeToNodes.get(end) === Infinity) return null;

    return getShortestPath(start, end, timeToNodes);

}

// Побудова графа за варіантом (8)

const graph = new Map();

for (let i = 1; i <= 6; i++) {

  graph.set(i, new Node(i));

}

function connect(a, b, weight) {

  graph.get(a).edges.push(new Edge(graph.get(b), weight));

  graph.get(b).edges.push(new Edge(graph.get(a), weight));

}

connect(1, 2, 2);

connect(1, 3, 5);

connect(1, 5, 2);

connect(2, 4, 7);

connect(2, 5, 1);

connect(2, 6, 8);

connect(3, 6, 5);

connect(3, 2, 8);

connect(4, 5, 1);

connect(5, 6, 3);

// Тест

const startNode = graph.get(2);

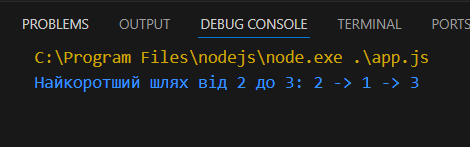
const endNode = graph.get(3);

const resultPath = getShortestPathMain(graph, startNode, endNode);

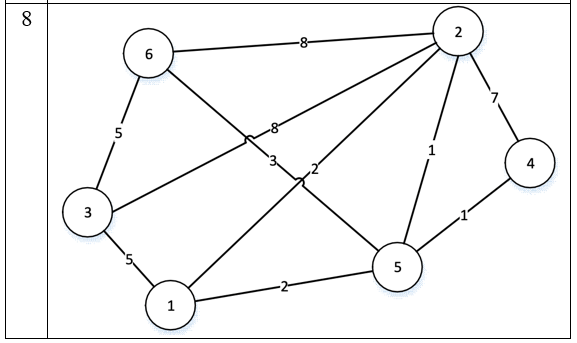
const pathIds = resultPath.map(node => node.id);

console.log(`Найкоротший шлях від ${startNode.id} до ${endNode.id}: ${pathIds.join(" -> ")}`);

Результат:



Перевірка:



Всі шляхи від 2 до 3:  
2 – 6 – 3 = 13

2 – 3 = 8

2 – 1 – 3 = 7

2 – 5 – 1 – 3 = 8

2 – 4 – 5 – 1 – 3 = 15

2 – 5 – 6 – 3 = 9

Шлях 2 – 1 – 3 дійсно найкоротший.

Код програми запушено на гіт-хаб в репозиторій https://github.com/DelovSlava/Algoritmi\_-\_Strutures

**Висновок:**

У ході виконання завдання було реалізовано симетричний (in-order) обхід бінарного дерева.

* реалізовано завдання із заміною значення вузлів, які складаються з однакових цифр, на 0. Перевірка виконувалась під час симетричного обходу дерева.
* Знайдено та виведено ті вузли, у яких різниця сум елементів правого і лівого піддерев перевищує 20. Для оптимізації було попередньо обчислено суму піддерева кожного вузла, що дозволило зменшити кількість обчислень і підвищити ефективність програми до O(n) замість O(n^2)

Робота демонструє ефективне використання рекурсії, оптимізацію підрахунків та правильне застосування властивостей бінарного дерева.