**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**Практична робота № 7**

**Тема:** Алгоритми на Графах. Алгоритм Дейкстри.

Виконав студент І курсу

Спеціальності «Інженерія

програмного забезпечення»

Боднарчук Роман Романович

**Ужгород-2025**

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

**Завдання до 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.

# Хід роботи

**Варіант 1**

|  |  |
| --- | --- |
| **№** | **Завдання** |
| 1 |  |

function dijkstra(graph, start, end) {

  let shortestDistances = {};

  let parents = {};

  let unvisitedNodes = Object.keys(graph);

  for(let node of unvisitedNodes) {

    shortestDistances[node] = Infinity;

  }

  shortestDistances[start] = 0;

  while(unvisitedNodes.length > 0) {

    let closestNode = null;

    for(let node of unvisitedNodes) {

      if(closestNode === null || shortestDistances[node] < shortestDistances[closestNode]) {

        closestNode = node;

      }

    }

    let currentNodeDist = shortestDistances[closestNode];

    let currentNeighbors = graph[closestNode];

    for(let neighbor in currentNeighbors) {

      let weight = currentNeighbors[neighbor];

      let oldDist = shortestDistances[neighbor];

      let predictedDist = currentNodeDist + weight;

      if(predictedDist < oldDist) {

        shortestDistances[neighbor] = predictedDist;

        parents[neighbor] = closestNode;

      }

    }

    unvisitedNodes = unvisitedNodes.filter(node => node !== closestNode);

  }

  let shortestPath = [end];

  let parent = parents[end];

  while(parent) {

    shortestPath.push(parent);

    parent = parents[parent];

  }

  let pathWithWeights = [];

  for(let i = 0; i < shortestPath.length - 1; i++) {

    let node1 = shortestPath[i];

    let node2 = shortestPath[i+1];

    let weight = graph[node2][node1];

    pathWithWeights.push(`${node1} -(${weight})->`);

  }

  pathWithWeights.push(start);

  return {

    distance: shortestDistances[end],

    path: pathWithWeights.join(' ')

  };

}

const graph = {

  1: {2: 5, 6: 1},

  2: {1: 5, 4: 0, 3: 5 },

  3: {2: 5, 5: 6},

  4: {6: 7, 2: 0, 5: 0 },

  5: {4: 0, 6: 4, 3: 6 },

  6: {1: 1, 4: 7, 5: 4},

};

const result1 = dijkstra(graph, '1', '5');

console.log(`Шлях від 5 до 1: ${result1.distance}`);

console.log(`Пройдений шлях: ${result1.path}`);

const result2 = dijkstra(graph, '3', '5');

console.log(`Шлях від 5 до 3: ${result2.distance}`);

console.log(`Пройдений шлях: ${result2.path}`);

const result3 = dijkstra(graph, '3', '6');

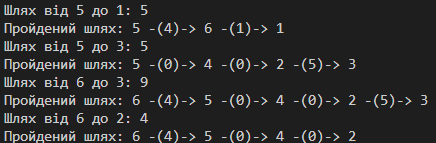
console.log(`Шлях від 6 до 3: ${result3.distance}`);

console.log(`Пройдений шлях: ${result3.path}`);

const result4 = dijkstra(graph, '2', '6');

console.log(`Шлях від 6 до 2: ${result4.distance}`);

console.log(`Пройдений шлях: ${result4.path}`);



**Питання для самоконтролю**

1. Що таке граф?

**Відповідь:** Граф - це математична структура, що складається з вершин (вузлів) і зв'язків (ребер), що з'єднують ці вершини.

1. Які є види графів ?

**Відповідь:** Існує кілька видів графів, серед найпоширеніших:

* **Напрямлені та ненапрямлені графи:** у напрямленому графі зв'язки мають напрямок, у ненапрямленому - ні.
* **Ваговані та неваговані графи:** вагований граф має числові значення (ваги) на кожному зв'язку, невагований - ні.
* **Планарні та непланарні графи:** планарний граф може бути намальований на площині без перетинів зв'язків, непланарний - ні.

1. Як можуть задаватися графи?

**Відповідь:** Графи можна задавати двома основними способами:

* **Матрицею суміжності**, де рядки і стовпці відповідають вершинам, а значення у комірках показують наявність (або вагу) зв'язку між вершинами.
* **Списком суміжності**, де кожній вершині відповідає список вершин, з якими вона має зв'язок.

1. Що таке вершина у графі?

**Відповідь:** Вершина у графі - це один з елементів структури графа, який може мати зв'язки з іншими вершинами.

1. Що таке ребро у графі?

**Відповідь:** Ребро у графі - це зв'язок між двома вершинами, вказує на те, що ці вершини пов'язані або сусідні у графі.

1. Які алгоритми для пошуку мінімальної відстані у графі вам відомі?

**Відповідь:** Найвідоміші алгоритми для пошуку мінімальної відстані у графі:

* **Алгоритм Дейкстри** - знаходить найкоротший шлях від однієї вершини до всіх інших у направленому або ненаправленому зваженому графі з не від'ємними вагами ребер.
* **Алгоритм Беллмана-Форда** - знаходить найкоротший шлях від однієї вершини до всіх інших у направленому зваженому графі, може працювати з ребрами з від'ємною вагою, але уникати циклів з від'ємною сумарною вагою.
* **Алгоритм Флойда-Уоршелла** - знаходить найкоротші шляхи між усіма парами вершин у зваженому графі (може мати від'ємні ваги ребер, але не може мати циклів з від'ємною сумарною вагою).

**Висновки:** У ході виконання Практичної роботи №7, я написав алгоритм Дейкстри. Алгоритм Дейкстри знаходить найкоротший шлях від одного вузла до всіх інших у графі з не від'ємними вагами ребер. Він працює шляхом поступового оновлення найкоротших відстаней до сусідніх вузлів через поточний вузол. Тим самим набув практичних навичок із виконанням алгоритмів Дейкстра та зробив висновок.