**Державний вищий навчальний заклад**

**Ужгородський національний університет**

**Факультет інформаційних технологій**

**Практична робота №7**

**Алгоритми на Графах. Алгоритм Дейкстри**

Виконав студент 1 курсу

спеціальності “Інженерія

програмного забезпечення”

Гарабаджіу Крістіан

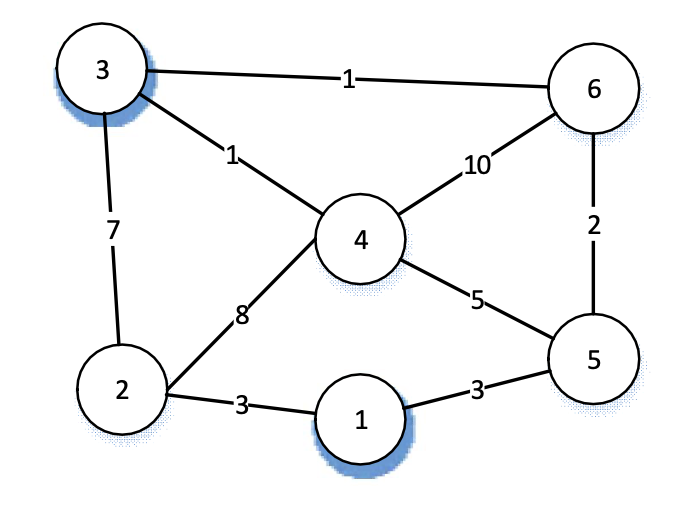
**Ужгород-2025**

**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:**

Граф:  


function dijkstra(reg = 1) {

const graph = {

"1": { "2": 3, "5": 3 },

"2": { "1": 3, "3": 7, "4": 8 },

"3": { "2": 7, "4": 1, "6": 1 },

"4": { "2": 8, "3": 1, "5": 5, "6": 10 },

"5": { "1": 3, "4": 5, "6": 2 },

"6": { "3": 1, "4": 10, "5": 2 }

};

if(reg == 1){var start = "1"; var end = "3";}

else if(reg == 2){var start = "3"; var end = "1";}

else{

return "error";

}

let distance = Infinity;

const visited = new Set();

const queue = new Map();

queue.set(start, 0);

while (queue.size > 0) {

const current = [...queue.entries()].reduce((a, b) => (a[1] < b[1] ? a : b))[0];

const currentDist = queue.get(current);

queue.delete(current);

visited.add(current);

for (let neighbor in graph[current]) {

if (visited.has(neighbor)) continue;

const newDist = currentDist + graph[current][neighbor];

if (neighbor === end) {

distance = newDist < distance ? newDist : distance;

}

const neighborDist = queue.has(neighbor) ? queue.get(neighbor) : Infinity;

if (newDist < neighborDist) {

queue.set(neighbor, newDist);

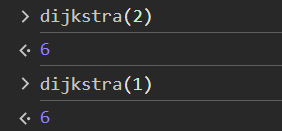
}

}

}

return distance;

}



**Обгрунтування ходу програми**

1. **Оголошення графа.** Створено об’єкт graph, який представляє граф через властивості типу “вершина - зв’язки”. Кожна вершина має свій відповідний об’єкт: список суміжних вершин із вагою ребра між ними.
2. **Вибір початкової та кінцевої вершини:**

* Якщо reg == 1, початкова вершина — “1”, кінцева — “3”.
* Якщо reg == 2, навпаки: початкова — “3”, кінцева — “1”.
* Якщо reg має інше значення — повертається “error”.

Зроблено це виключно для перевірки на помилки

1. **Ініціалізація змінних. D**istance — змінна, яка тримає мінімальну знайдену відстань між стартом і кінцем (спочатку Infinity). Visited — множина відвіданих вершин. Queue — Map, де зберігаються пари: вершина та відстань до неї. Використовується як черга з пріоритетом (за допомогою reduce і компараторів)
2. **Додавання стартової вершини в чергу з відстанню 0**
3. **Основний цикл.** Поки черга не пуста:

* Обирається вершина з найменшою відстанню (current) з черги.
* Надалі ми цю вершину “обробляємо”, вона позначається позначається як відвідана і видаляється з черги.
* Переглядаються всі її сусіди:
* Якщо сусід вже відвіданий — пропускається.
* Обчислюється нова потенційна відстань до сусіда (newDist).
* Якщо сусід є кінцевою вершиною (end) — перевіряється, чи нова відстань коротша за збережену у distance, і за потреби оновлюється.
* Якщо newDist менша, ніж попередня збережена відстань до цього сусіда — вона оновлюється в черзі.

1. **Результат:** Після завершення циклу повертається змінна distance, яка містить найкоротшу знайдену відстань від старту до фінішу.

**Висновок:**

У результаті виконання роботи було успішно реалізовано алгоритм Дейкстри для знаходження найкоротшого шляху між двома вершинами графа. Граф задано у вигляді об’єкта, де кожна вершина зберігає список своїх сусідів із відповідними вагами ребер. Для збереження відстаней використовується Map, що забезпечує швидкий доступ до поточних значень і можливість оновлення. В процесі пошуку вершина з найменшою відстанню вибирається за допомогою функції reduce, після чого розглядаються всі її сусіди, і, за потреби, оновлюються відстані до них. Якщо сусід є кінцевою вершиною (3 або 1), оновлюється мінімальна знайдена відстань. Алгоритм завершується поверненням найкоротшої відстані до кінцевої точки, демонструючи правильність і ефективність реалізованого підходу.