**Прізвище:** Дацишин

**Ім’я:** Роман

**Група:** КН-405

**Варіант:** 4

**Кафедра.:** Кафедра Систем

Автоматизованого Проектування

**Дисципліна:** Дискретні моделі в САПР

**Перевірив:** Кривий Р.З.

**Звіт**

До лабораторної роботи №1

На тему “Побудова мінімального остового дерева”

**Мета роботи:** Метою даної лабораторної роботи є вивчення алгоритмів рішення задач побудови остових дерев.

**Короткі теоретичні відомості:**

Графом G називають скінчену множину V з нерефлексивним симетричним відношенням R на V. Визначим E як множину симетричних пар в R. Кожний елемент V називають вершиною. Кожний елемент Е називають ребром, а E множиною ребер G.

Граф називається зв’язним, якщо в ньому для будь-якої пари вершин знайдеться ланцюг, який їх з’єднує, тобто, якщо по ребрах (дугах) можна попасти з будь-якої вершини в іншу.

Цикл - це ланцюг, в якого початкова і кінцева точки співпадають.

Дерево - це зв’язний граф без циклів.

Алгоритм Борувки.

Це алгоритм знаходження мінімального остового дерева в графі. Вперше був опублікований в 1926 році Отакаром Борувкой, як метод знаходження оптимальної електричної мережі в Моравії. Робота алгоритму складається з декількох ітерацій, кожна з яких полягає в послідовному додаванні ребер до остового лісу графа, до тих пір, поки ліс не перетвориться на дерево, тобто, ліс, що складається з однієї компоненти зв'язності.

У псевдокоді, алгоритм можна описати так:

1. Спочатку, нехай T - порожня множина ребер (представляє собою остовий ліс, до якого кожна вершина входить в якості окремого дерева).

2. Поки T не є деревом (поки число ребер у T менше, ніж V-1, де V - кількість вершин у графі):

a. Для кожної компоненти зв'язності (тобто, дерева в остовому лісі) в підпункті з ребрами T, знайдемо ребро найменшої ваги, що зв'язує цю компоненту з деякої іншої компонентою зв'язності. (Передбачається, що ваги ребер різні, або як-то додатково впорядковані так, щоб завжди можна було знайти єдине ребро з мінімальною вагою).

b. Додамо всі знайдені ребра в множину T.

3. Отримана множина ребер T є мінімальним остовим деревом вхідного графа.

**Індивідуальне завдання:**

Студенти, які в загальному списку, мають порядковий номер від 1 до 10 - реалізовують Алгоритм Прим.

Вхідні дані:

8

0 3 0 0 0 34 0 80

3 0 0 1 0 0 0 68

0 0 0 0 23 0 12 0

0 1 0 0 53 0 0 39

0 0 23 53 0 0 68 14

34 0 0 0 0 0 0 25

0 0 12 0 68 0 0 99

80 68 0 39 14 25 99 0

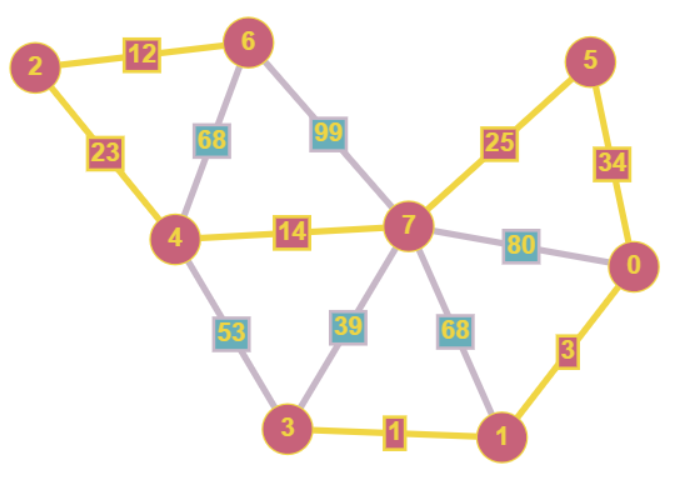


Рис. 1 Візуальне представлення графа, а також застосований алгоритм Прим

Було написано програму для знаходження остового дерева методом Прим.

Код програми , а також результат її виконання наведено нижче

**Код програми**

**let matrixLength;**

**let initialString = "";**

**let html = "";**

**let m1 = [];**

**const MAX\_INTEGER = Number.MAX\_SAFE\_INTEGER;**

**const MIN\_INTEGER = Number.MIN\_SAFE\_INTEGER;**

**let input = document.querySelector("input");**

**input.addEventListener("change", () => {**

**let files = input.files;**

**if (files.length == 0) return;**

**const file = files[0];**

**let reader = new FileReader();**

**reader.onload = (e) => {**

**let file = e.target.result;**

**file = file.replace(/(\r\n|\n|\r)/gm, " ");**

**initialString = file;**

**parseMatrix(initialString);**

**};**

**reader.onerror = (e) => alert(e.target.error.name);**

**reader.readAsText(file);**

**});**

**const minValue = (key, mstSet) => {**

**let min = MAX\_INTEGER;**

**let minIndex = MIN\_INTEGER;**

**for (let i = 0; i < matrixLength; i++) {**

**if (mstSet[i] === false && key[i] < min) {**

**min = key[i];**

**minIndex = i;**

**}**

**}**

**return minIndex;**

**};**

**const printMST = (parent, graph) => {**

**html += `<br/>Edge &#8195;Weight <br/>`;**

**console.log("Edge \tWeight");**

**for (let i = 1; i < matrixLength; i++) {**

**html += `${parent[i]} - ${i} &#8195; ${graph[i][parent[i]]} <br/>`;**

**console.log(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);**

**}**

**};**

**const primMST = (graph) => {**

**let parent = [];**

**let key = [];**

**let mstSet = [];**

**for (let i = 0; i < matrixLength; i++) {**

**key[i] = MAX\_INTEGER;**

**mstSet[i] = false;**

**}**

**key[0] = 0;**

**parent[0] = -1;**

**for (let i = 0; i < matrixLength - 1; i++) {**

**let u = minValue(key, mstSet);**

**mstSet[u] = true;**

**for (let j = 0; j < matrixLength; j++) {**

**if (graph[u][j] !== 0 && mstSet[j] === false && graph[u][j] < key[j]) {**

**parent[j] = u;**

**key[j] = graph[u][j];**

**}**

**}**

**}**

**printMST(parent, graph);**

**document.getElementById("container").innerHTML = html;**

**};**

**const parseMatrix = (matrix) => {**

**matrixLength = matrix[0];**

**let str = matrix.substring(2);**

**let arr = [];**

**let num = "";**

**for (let i = 0; i < str.length; i++) {**

**if (str[i] !== " ") {**

**num += str[i];**

**} else {**

**arr.push(parseInt(num));**

**num = "";**

**}**

**}**

**let m = [];**

**for (let i = 0; i < matrixLength; i++) {**

**m[i] = new Array(matrixLength);**

**}**

**let j = 0;**

**let k = 0;**

**for (let i = 0; i < arr.length; i++) {**

**if (j < Math.sqrt(arr.length)) {**

**m[k][j] = arr[i];**

**j++;**

**} else {**

**j = 0;**

**k++;**

**m[k][j] = arr[i];**

**j++;**

**}**

**}**

**m1 = m;**

**primMST(m1);**

**};**

****

Рис. 2. Результат виконання програми

Отже, результат виконання програми збігається з вище отриманими графічними розрахунками.

Повна версія коду доступна на GitHub: <https://github.com/RomanDatsyshyn/DiscreteModelsLabs>

**Висновок:** під час виконання цієї лабораторної роботи я ознайомився з алгоритмами рішення задач побудови остових дерев та застосував ці знання на практиці, а саме написав програму для знаходження остового дерева методом Прим, та порівняв отримані результати з графічними.