ЗВІТ

про виконання лабораторної роботи № 4

Комбінаторний аналіз.

з дисципліни «Програмування дискретних структур»

студента групи КН-2226Б

Медвідь Дмитро Степанович

Варіант 11

Завдання №1

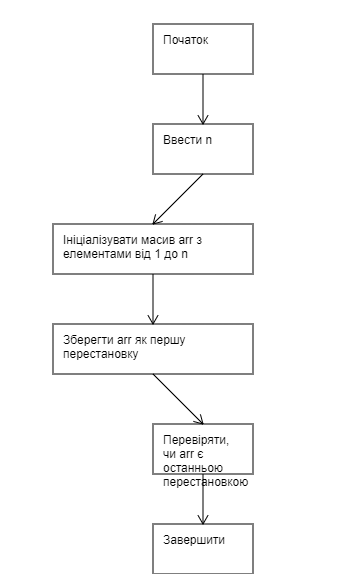
1. Умова завдання;

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано натуральне число n. Навести в лексикографічному порядку всі

перестановки елементів множини {1, 2, ... , n}.

1. опис алгоритму (блок-схема або структурна схема);



1. текст програми;

// Функція для знаходження наступної перестановки

function nextPermutation(arr) {

    // Знаходимо найбільше і, для якого arr[i] < arr[i+1]

    let i = arr.length - 2;

    while (i >= 0 && arr[i] >= arr[i + 1]) {

        i--;

    }

    // Якщо такого і немає, то перестановки завершені

    if (i >= 0) {

        // Знаходимо найбільше j, для якого arr[j] > arr[i]

        let j = arr.length - 1;

        while (arr[j] <= arr[i]) {

            j--;

        }

        // Міняємо місцями arr[i] та arr[j]

        [arr[i], arr[j]] = [arr[j], arr[i]];

    }

    // Перевертаємо підмасив arr[i+1...n] для отримання наступної перестановки

    reverse(arr, i + 1);

}

// Функція для перевертання масиву з позиції start до кінця

function reverse(arr, start) {

    let end = arr.length - 1;

    while (start < end) {

        [arr[start], arr[end]] = [arr[end], arr[start]];

        start++;

        end--;

    }

}

// Головна функція для знаходження всіх перестановок

function permutations(n) {

    // Створюємо початкову перестановку [1, 2, ..., n]

    let arr = Array.from({ length: n }, (\_, i) => i + 1);

    let result = [arr.slice()]; // Зберігаємо першу перестановку

    while (true) {

        nextPermutation(arr); // Знаходимо наступну перестановку

        // Якщо досягли початкової перестановки, то завершуємо

        if (arr.join('') === Array.from({ length: n }, (\_, i) => i + 1).join('')) {

            break;

        }

        result.push(arr.slice()); // Додаємо перестановку до результату

    }

    return result;

}

let n = 3;

let result = permutations(n);

console.log(result.map(p => p.join(' '))); // Виводимо всі перестановки

module.exports = { permutations };

1. значення вхідних даних та результати виконання програми;



1. посилання на публічний репозиторій GitHub з кодом програми;

https://github.com/DartVeyder/Discrete-programming/blob/main/lab4/task1.js

1. результати unit-тесту

const { permutations } = require('./task1');

describe('permutations', () => {

    test('permutations of 3 elements', () => {

        const result = permutations(3);

        expect(result).toEqual([

            [1, 2, 3],

            [1, 3, 2],

            [2, 1, 3],

            [2, 3, 1],

            [3, 1, 2],

            [3, 2, 1]

        ]);

    });

    test('permutations of 4 elements', () => {

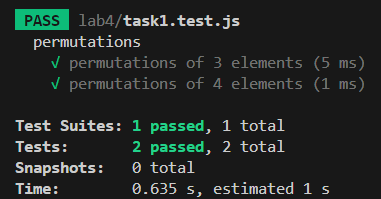
        const result = permutations(4);

        expect(result).toHaveLength(24); // 4!

        // You can add more specific expectations if needed

    });

});



Завдання №2

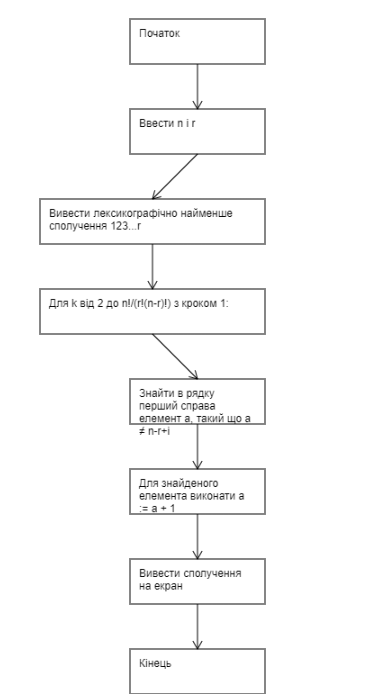
1. Умова завдання;

Задано натуральне число n і невід’ємне ціле число r (r ≤ n). Навести в

лексикографічному порядку всі r-сполучення без повторень з елементів множини

{1, 2, ... , n}.

1. Опис алгоритму (блок-схема або структурна схема);



1. текст програми;

// Функція для знаходження наступного сполучення

function nextCombination(n, r, arr) {

    let i = r - 1;

    // Знаходимо перший зліва елемент, який можна збільшити

    while (i >= 0 && arr[i] === n - r + i + 1) {

        i--;

    }

    // Якщо такого елемента немає, то сполучення завершені

    if (i === -1) {

        return false;

    }

    arr[i]++; // Збільшуємо цей елемент

    // Заповнюємо решту елементів послідовністю

    for (let j = i + 1; j < r; j++) {

        arr[j] = arr[j - 1] + 1;

    }

    return true;

}

// Головна функція для знаходження всіх сполучень

function combinations(n, r) {

    let arr = Array.from({ length: r }, (\_, i) => i + 1); // Початкове сполучення [1, 2, ..., r]

    let result = [arr.slice()]; // Зберігаємо перше сполучення

    while (true) {

        if (!nextCombination(n, r, arr)) { // Знаходимо наступне сполучення

            break;

        }

        result.push(arr.slice()); // Додаємо сполучення до результату

    }

    return result;

}

let n = 5;

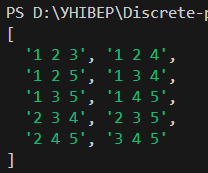
let r = 3;

let result = combinations(n, r);

console.log(result.map(c => c.join(' '))); // Виводимо всі сполучення

module.exports = { combinations };

1. значення вхідних даних та результати виконання програми;



1. посилання на публічний репозиторій GitHub з кодом програми;

https://github.com/DartVeyder/Discrete-programming/blob/main/lab4/task2.js

1. результати unit-тесту

// combinations.test.js

const { combinations } = require('./task2');

describe('combinations', () => {

    test('combinations of 5 choose 3', () => {

        const result = combinations(5, 3);

        expect(result).toEqual([

            [1, 2, 3],

            [1, 2, 4],

            [1, 2, 5],

            [1, 3, 4],

            [1, 3, 5],

            [1, 4, 5],

            [2, 3, 4],

            [2, 3, 5],

            [2, 4, 5],

            [3, 4, 5]

        ]);

    });

    test('combinations of 4 choose 2', () => {

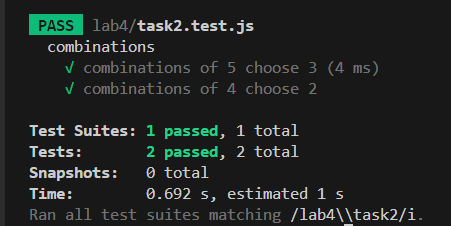
        const result = combinations(4, 2);

        expect(result).toHaveLength(6); // 4 choose 2

        // You can add more specific expectations if needed

    });

});



Висновок

Поглибив і закріпив розуміння основних понять комбінаторного аналізу,

формувати навички програмування алгоритмів генерування перестановок та

сполучень, обчислення розв’язків рекурентних рівнянь.