**Державний вищий навчальний заклад  
Ужгородський національний університет  
Факультет інформаційних технологій**

ПРАКТИЧНА РОБОТА № 6  
**Тема:** Алгоритми на деревах

Виконав студент  
І курсу спеціальності «Інженерія програмного забезпечення»

Романюк Артем

**Ужгород-2025**

**Мета:** набути навичок створення та обробки дерев.

**Питання для самоконтролю**

1. З чим пов’язана популярність використання дерев у програмуванні?
2. Чи можна список віднести до дерев? Відповідь обґрунтуйте.
3. Які дані містять адресні поля елемента бінарного дерева?
4. Чи може бінарне дерево бути строгим і неповним? Відповідь обґрунтуйте.
5. Чи може бінарне дерево бути нестрогим і повним? Відповідь обґрунтуйте.
6. Яким може бути майже збалансоване бінарне дерево: повним, неповним, строгим, нестрогим? Відповідь обґрунтуйте.
7. Куди може бути доданий елемент у бінарне дерево у залежності від його виду (повне, неповне, строгим, нестрогим)? Вид дерева при цьому повинен зберегтися.
8. Куди може бути доданий елемент у збалансоване бінарне дерево? Вид дерева при цьому повинен зберегтися.
9. Чим відрізняються, з точки зору реалізації алгоритму прямий, симетричний і зворотний обходи бінарного дерева?

Завдання до роботи:

1. Написати код програми відповідно до обраного варіанту. Кожен варіант містить завдання та спосіб обходу дерева для кожного із завдань.
2. Оформити звіт та завантажити його в системі електронного навчання ДВНЗ «УжНУ» в установлений термін.
3. Підготувати відповіді на контрольні питання.

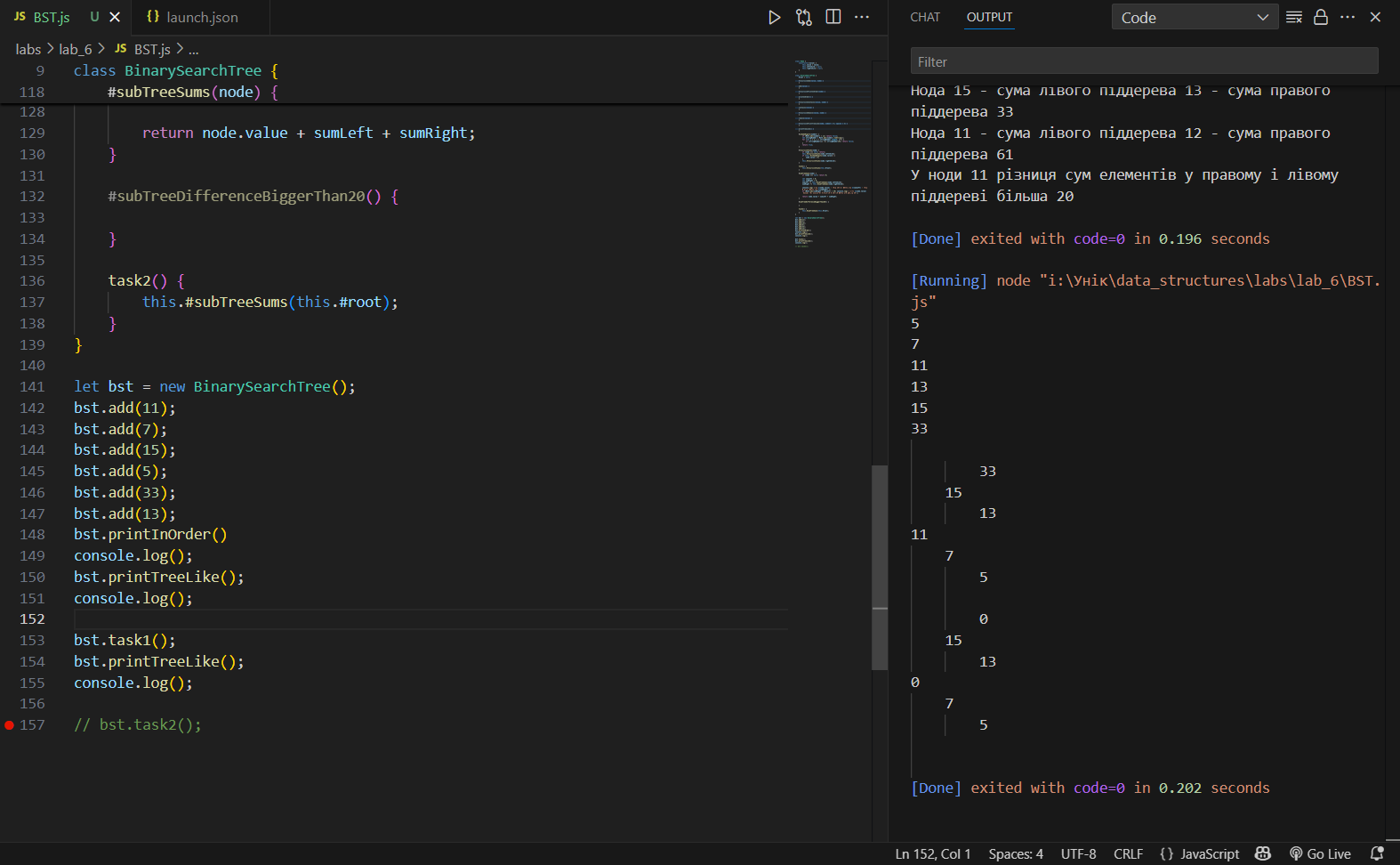
**Хід роботи:**

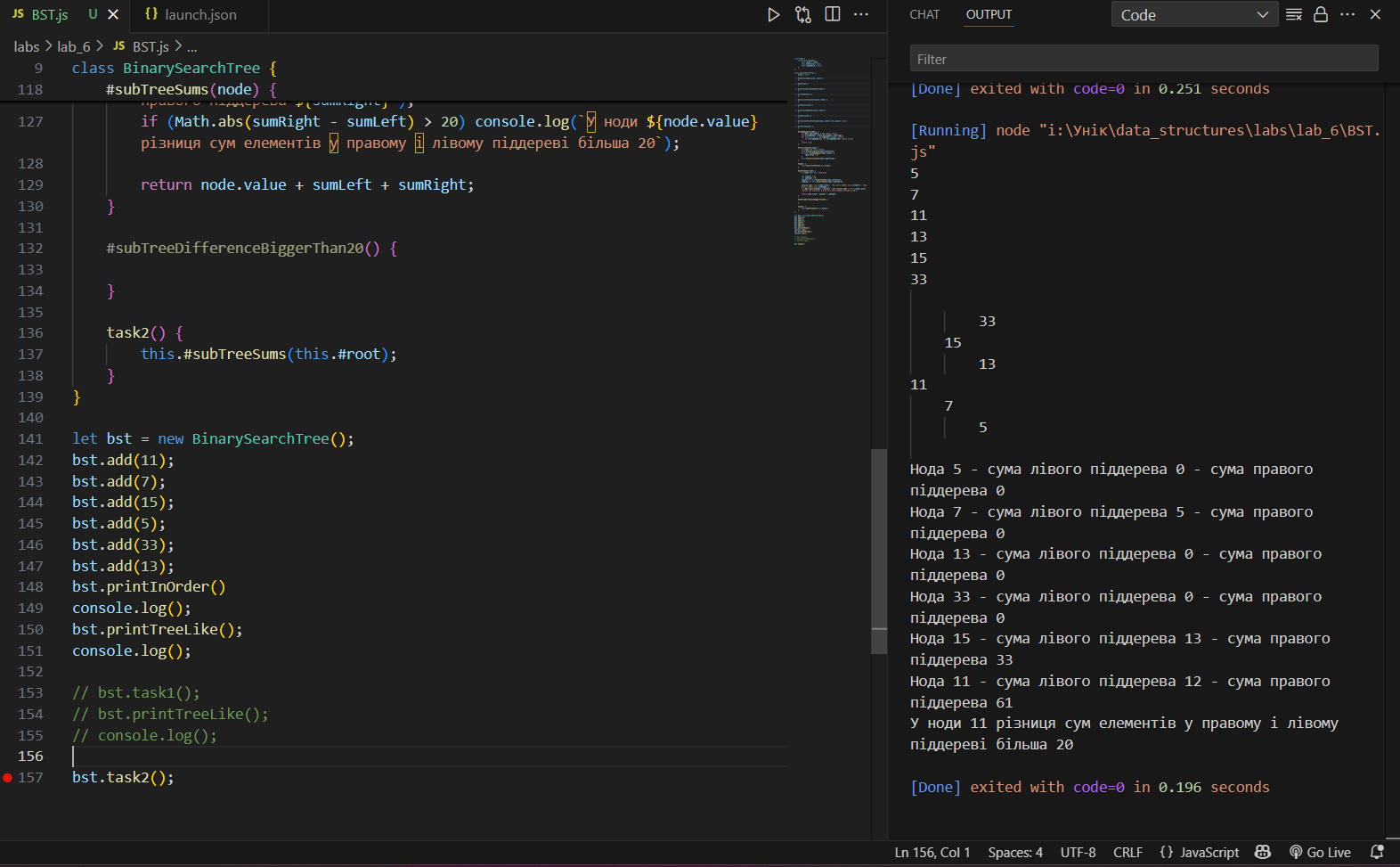
**Варіант 27 (9)**

| 9 | Симетричний | 1. Ті елементи дерева, які складаються з однакових цифр, замінити значенням 0  2. Вивести на екран ті вузли, у яких різниця сум елементів в правому і лівому піддереві більша 20. Вивести ці суми для кожного вузла |
| --- | --- | --- |

Реалізація в коді:

| class Node {  constructor(value) {  this.value = value;  this.leftChild = null;  this.rightChild = null;  }  }  class BinarySearchTree {  #root = null;  #recursiveAdd(value, node) {  if (node === null) return new Node(value);  if (value < node.value) {  node.leftChild = this.#recursiveAdd(value, node.leftChild);  } else if (value > node.value) {  node.rightChild = this.#recursiveAdd(value, node.rightChild);  }  return node;  }  add(value) {  this.#root = this.#recursiveAdd(value, this.#root);  }  #recursivePrintInOrder(node) {  if (node === null) return;  this.#recursivePrintInOrder(node.leftChild);  console.log(node.value);  this.#recursivePrintInOrder(node.rightChild);  }  printInOrder() {  this.#recursivePrintInOrder(this.#root);  }  #recursiveContains(value, node) {  if (node === null) return false;  if (value === node.value) return true;  else if (value < node.value) return this.#recursiveContains(value, node.leftChild);  else if (value > node.value) return this.#recursiveContains(value, node.rightChild);  }  contains(value) {  return this.#recursiveContains(value, this.#root);  }  #recursiveRemove(value, node) {  if (node === null) return null;  if (value < node.value) node.leftChild = this.#recursiveRemove(value, node.leftChild);  else if (value > node.value) node.rightChild = this.#recursiveRemove(value, node.rightChild);  else {  if (node.leftChild === null && node.rightChild === null) {  return null;  } else if (node.leftChild !== null && node.rightChild !== null) {  let swapNode = node.rightChild;  let parent = node;  while (swapNode.leftChild) {  parent = swapNode;  swapNode = swapNode.leftChild;  }  if (parent === node) {  parent.rightChild = swapNode.rightChild;  } else {  parent.leftChild = swapNode.rightChild;  }  node.value = swapNode.value;  return node;  } else {  if (node.leftChild !== null) {  return node.leftChild;  } else {  return node.rightChild;  }  }  }  return node;  }  remove(value) {  this.#root = this.#recursiveRemove(value, this.#root);  }  #recursivePrintTreeLike(node, indent = 0, spaces = 4) {  if (node === null) return;  indent += spaces;  this.#recursivePrintTreeLike(node.rightChild, indent);  console.log(" ".repeat(indent - spaces) + node.value);  this.#recursivePrintTreeLike(node.leftChild, indent);  }  printTreeLike() {  this.#recursivePrintTreeLike(this.#root);  }  #isSameDigits(number) {  if (Math.abs(number) < 11) return false;  let stringNumber = Math.abs(number).toString();  for (let i = 1; i < stringNumber.length; i++) {  if (stringNumber[i] !== stringNumber[0]) return false;  }  return true;  }  #recursiveTask1(node) {  if (node === null) return;  this.#recursiveTask1(node.leftChild);  if (this.#isSameDigits(node.value)) {  node.value = 0;  }  this.#recursiveTask1(node.rightChild);  }  task1() {  this.#recursiveTask1(this.#root);  }  #subTreeSums(node) {  if (node === null) return 0;  let sumLeft = 0;  let sumRight = 0;  sumLeft += this.#subTreeSums(node.leftChild);  sumRight += this.#subTreeSums(node.rightChild);  console.log(`Нода ${node.value} - сума лівого піддерева ${sumLeft} - сума правого піддерева ${sumRight}`);  if (Math.abs(sumRight - sumLeft) > 20) console.log(`У ноди ${node.value} різниця сум елементів у правому і лівому піддереві більша 20`);  return node.value + sumLeft + sumRight;  }  task2() {  this.#subTreeSums(this.#root);  }  }  let bst = new BinarySearchTree();  bst.add(11);  bst.add(7);  bst.add(15);  bst.add(5);  bst.add(33);  bst.add(13);  bst.printInOrder()  console.log();  bst.printTreeLike();  console.log();  bst.task1();  bst.printTreeLike();  console.log();  // bst.task2(); |
| --- |





**Висновок:**

У ході виконання практичної роботи було закріплено теоретичні знання про бінарні дерева та набуті практичні навички з їх реалізації на мові програмування JavaScript. Було реалізовано структуру бінарного дерева пошуку з можливістю додавання, видалення та виведення вузлів, а також виконано два завдання: заміна значень елементів, що складаються з однакових цифр, на 0, і знаходження вузлів, у яких різниця сум значень у піддеревах перевищує 20. Завдяки цій роботі було покращено розуміння принципів роботи з деревоподібними структурами даних і алгоритмами їх обходу.