Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Report on '**Lab Work 3'** [COMP 314]

Submitted by:

Chandan Kumar Mahato (31)

III-year, II semester

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: Apr 23, 2023

Qlab1

Implementation, testing of binary search tree

Solution:

Implementing BinarySearchTree algorithm using JavaScript (node.js) and writing test using (jest) a JavaScript testing framework.

Source Code:

https://github.com/ChandankMahato/DSA_Lab_6th_Sem

Scripts:

npm start
npm run test

1. **BinarySearchTree**

```
class Node {
  constructor(key, value) {
    this.key = key;
    this.value = value;
    this.left = null;
    this.right = null;
  }
}

class BinarySearchTree {
  constructor() {
    this.root = null;
    this.count = 0;
  }

  size() {
    return this.count;
  }

  add(key, value) {
    this.root = this._addNode(this.root, key, value);
    this.count++;
  }

  _addNode(node, key, value) {
    if (node === null) {
      return new Node(key, value);
    }
    if (key < node.key) {
      node.left = this._addNode(node.left, key, value);
    } else if (key > node.key) {
      node.right = this._addNode(node.right, key, value);
    } else {
      node.value = value;
    }
    return node;
  }

  search(key) {
    let node = this._searchNode(this.root, key);
    return node ? node.value : false;
  }
  _searchNode(node, key) {
    if (node === null || node.key === key) {
      return node;
    }
    if (key < node.key) {
      return this._searchNode(node.left, key);
    } else {
      return this._searchNode(node.right, key);
    }
  }

  smallest() {
    let node = this._smallestNode(this.root);
    return node ? [node.key, node.value] : null;
  }
  _smallestNode(node) {
    if (node === null || node.left === null) {
      return node;
    }
    return this._smallestNode(node.left);
  }

  largest() {
    let node = this._largestNode(this.root);
    return node ? [node.key, node.value] : null;
  }
  _largestNode(node) {
    if (node == null || node.right === null) {
      return node;
    }
    return this._largestNode(node.right);
  }

  remove(key) {
    let node = this._removeNode(this.root, key);
    if (node !== null) {
      this.count--;
      return true;
    } else {
      return false;
    }
  }
  _removeNode(node, key) {
    if (node === null) {
      return null;
    }

    if (key < node.key) {
      node.left = this._removeNode(node.left, key);
      return node;
    } else if (key > node.key) {
      node.right = this._removeNode(node.right, key);
      return node;
    } else {
      if (node.left === null && node.right === null) {
        node = null;
        return node;
      }

      if (node.left === null) {
        node = node.right;
        return node;
      }

      if (node.right === null) {
        node = node.left;
        return node;
      }

      let tempNode = this._smallestNode(node.right);
      node.key = tempNode.key;
      node.value = tempNode.value;
      node.right = this._removeNode(node.right, tempNode.key);
      return node;
    }
  }

  inorder() {
    const nodes = [];
    function traverse(node) {
      if (node !== null) {
        traverse(node.left);
        nodes.push(node.key);
        traverse(node.right);
      }
    }
    traverse(this.root);
    return nodes;
  }

  preorder() {
    const nodes = [];
    function traverse(node) {
      if (node !== null) {
        nodes.push(node.key);
        traverse(node.left);
        traverse(node.right);
      }
    }
    traverse(this.root);
    return nodes;
  }

  postorder() {
    const nodes = [];
    function traverse(node) {
      if (node !== null) {
        traverse(node.left);
        traverse(node.right);
        nodes.push(node.key);
      }
    }
    traverse(this.root);
    return nodes;
  }
}

module.exports = {
  BinarySearchTree,
};
```

Test Implementation:

```
const { BinarySearchTree } = require("../../BST");

describe("BinarySearchTree", () => {
  let bst;
  beforeEach(() => {
    bst = new BinarySearchTree();
    bst.add(10, "Value for 10");
    bst.add(52, "Value for 52");
    bst.add(5, "Value for 5");
    bst.add(8, "Value for 8");
    bst.add(1, "Value for 1");
    bst.add(40, "Value for 40");
    bst.add(30, "Value for 30");
    bst.add(45, "Value for 45");
  });
  test("add", () => {
    const bsTree = new BinarySearchTree();
    expect(bsTree.size()).toBe(0);

    bsTree.add(15, "Value for 15");
    expect(bsTree.size()).toBe(1);

    bsTree.add(10, "Value for 10");
    expect(bsTree.size()).toBe(2);

    expect(bsTree.search(10)).toBe("Value for 10");
    expect(bsTree.search(15)).toBe("Value for 15");
  });
  test("inorder", () => {
    const expectedOutput = [1, 5, 8, 10, 30, 40, 45, 52];
    expect(bst.inorder()).toEqual(expectedOutput);

    bst.add(25, "Value for 25");
    const newExpectedOutput = [1, 5, 8, 10, 25, 30, 40, 45, 52];
    expect(bst.inorder()).toEqual(newExpectedOutput);
  });
  test("postorder", () => {
    const expectedOutput = [1, 8, 5, 30, 45, 40, 52, 10];
    expect(bst.postorder()).toEqual(expectedOutput);

    bst.add(25, "Value for 25");
    const newExpectedOutput = [1, 8, 5, 25, 30, 45, 40, 52, 10];
    expect(bst.postorder()).toEqual(newExpectedOutput);
  });
  test("test preorder walk", () => {
    expect(bst.preorder()).toEqual([10, 5, 1, 8, 52, 40, 30, 45]);
    bst.add(25, "Value for 25");
    expect(bst.preorder()).toEqual([10, 5, 1, 8, 52, 40, 30, 25, 45]);
  });
  test("test search", () => {
    expect(bst.search(40)).toEqual("Value for 40");
    expect(bst.search(90)).toBeFalsy();
    bst.add(90, "Value for 90");
    expect(bst.search(90)).toEqual("Value for 90");
  });
  test("test remove", () => {
    bst.remove(40);
    expect(bst.size()).toEqual(7);
    expect(bst.inorder()).toEqual([1, 5, 8, 10, 30, 45, 52]);
    expect(bst.preorder()).toEqual([10, 5, 1, 8, 52, 45, 30]);
  });
  test("test smallest", () => {
    expect(bst.smallest()).toEqual([1, "Value for 1"]);
    bst.add(6, "Value for 6");
    bst.add(4, "Value for 4");
    bst.add(0, "Value for 0");
    bst.add(32, "Value for 32");
    expect(bst.smallest()).toEqual([0, "Value for 0"]);
  });
  test("test largest", () => {
    expect(bst.largest()).toEqual([52, "Value for 52"]);
    bst.add(6, "Value for 6");
    bst.add(54, "Value for 54");
    expect(bst.largest()).toEqual([54, "Value for 54"]);
  });
});
```

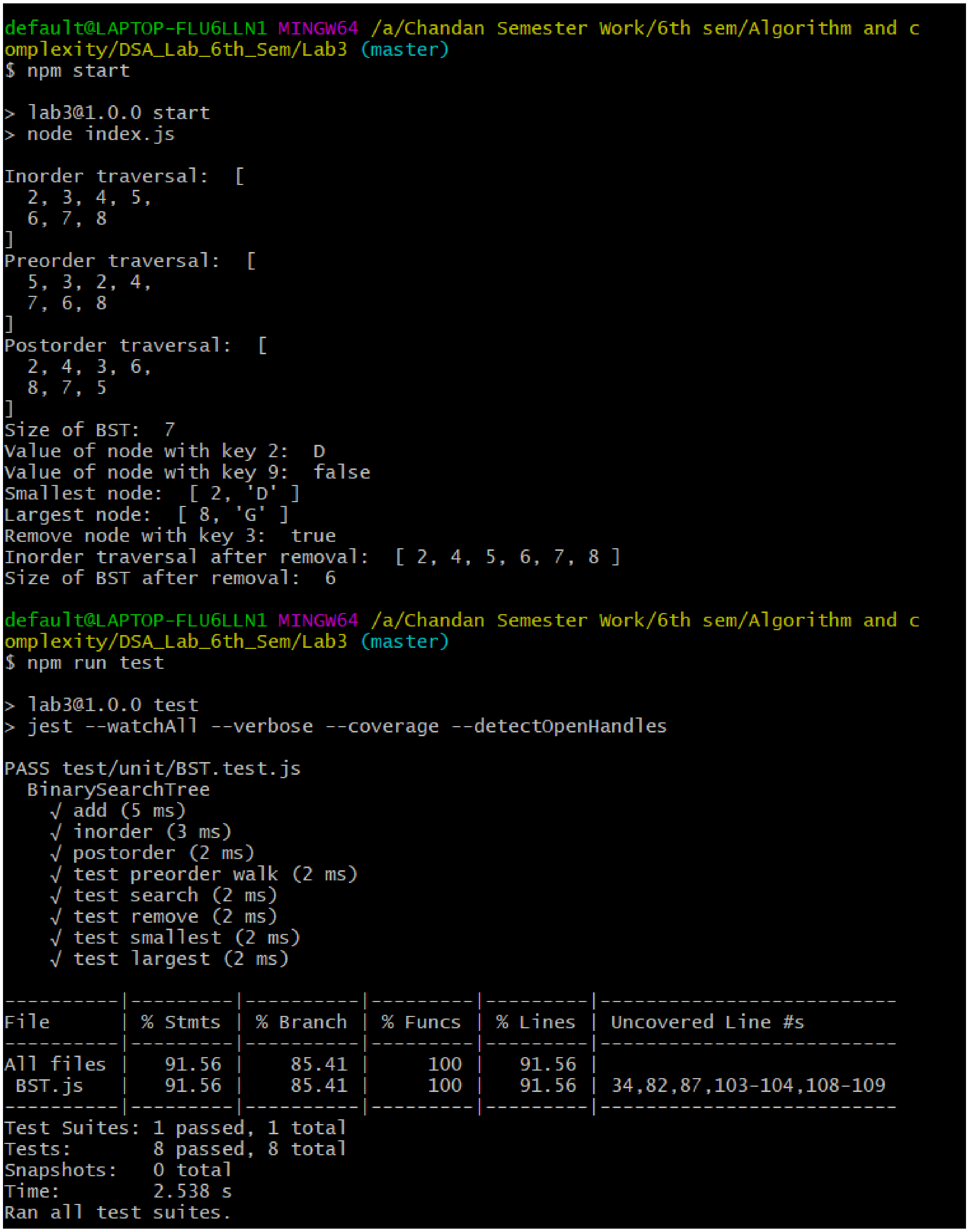
Observation:

Binary Search Tree is a data structure used for organizing and storing elements in a tree like structure. It is a type of binary tree where each node has at most two child nodes, and the left child is always less than the parent node, while the right child is always greater. This property makes it efficient for searching and sorting operations, as the elements can be quickly located using binary search. However, if the tree is not balanced, the worst-case time complexity for searching and inserting can be O(n), where n is the number of elements in the tree.

Conclusion:

BinarySearchTree were implemented in the JavaScript. The written code was benchmarked, and test cases were written using jest.

Output and Screenshots



```
default@LAPTOP-FLU6LLN1 MINGW64 /a/Chandan Semester Work/6th sem/Algorithm and c
omplexity/DSA_Lab_6th_Sem/Lab3 (master)
$ npm start

> lab3@1.0.0 start
> node index.js

Inorder traversal:  [
  2, 3, 4, 5,
  6, 7, 8
]
Preorder traversal:  [
  5, 3, 2, 4,
  7, 6, 8
]
Postorder traversal:  [
  2, 4, 3, 6,
  8, 7, 5
]
Size of BST:  7
Value of node with key 2:  D
Value of node with key 9:  false
Smallest node:  [ 2, 'D' ]
Largest node:  [ 8, 'G' ]
Remove node with key 3:  true
Inorder traversal after removal:  [ 2, 4, 5, 6, 7, 8 ]
Size of BST after removal:  6

default@LAPTOP-FLU6LLN1 MINGW64 /a/Chandan Semester Work/6th sem/Algorithm and c
omplexity/DSA_Lab_6th_Sem/Lab3 (master)
$ npm run test

> lab3@1.0.0 test
> jest --watchAll --verbose --coverage --detectOpenHandles

PASS test/unit/BST.test.js
  BinarySearchTree
    √ add (5 ms)
    √ inorder (3 ms)
    √ postorder (2 ms)
    √ test preorder walk (2 ms)
    √ test search (2 ms)
    √ test remove (2 ms)
    √ test smallest (2 ms)
    √ test largest (2 ms)

----------|---------|----------|---------|---------|-------------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------------
All files |   91.56 |    85.41 |     100 |   91.56 |
 BST.js   |   91.56 |    85.41 |     100 |   91.56 | 34,82,87,103-104,108-109
----------|---------|----------|---------|---------|-------------------------
Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.538 s
Ran all test suites.
```

Fig 1: Output and algorithm test case