100

Tamir Krief, Iaian Milton, Blessing Abumere
COSC 336
11/5/2024

Assignment 6

Exercise 1.

keys: 10, 22, 31, 4, 15, 28, 17, 88, 59
hash table length: m=11
hash function h(x) = x(mod 11)

Linear probing:
h(10) = 10 mod 11 = 10
h(22) = 22 mod 11 = 0
h(31) = 31 mod 11 = 9
h(4) = 4 mod 11 = 4
h(15) = 15 mod 11 = 4
h(28) = 28 mod 11 = 6
h(17) = 17 mod 11 = 6
h(88) = 88 mod 11 = 0
h(59) = 59 mod 11 = 4

inserting the values:
10 [- - - - - - - - - - 10]
22 [22 - - - - - - - - - 10]
31 [22 - - - - - - - - 31 10]
4 [22 - - - 4 - - - - 31 10]
15 [22 - - - 4 15 - - - 31 10]
28 [22 - - - 4 15 28 - - 31 10]
17 [22 - - - 4 15 28 17 - 31 10]
88 [22 - - - 4 15 28 17 88 31 10]
59 [22 88 - - 4 15 28 17 59 31 10]

final table: [22 88 - - 4 15 28 17 59 31 10]

Quadratic Probing:

10 [- - - - - - - - - - 10]
22 [22 - - - - - - - - - 10]
31 [22 - - - - - - - - 31 10]

4 [22 - - - 4 - - - - 31 10]
15 [22 - - - 4 15 - - - 31 10]
28 [22 - - - 4 15 28 - - 31 10]
17 [22 - - - 4 15 28 17 - 31 10]
88 [22 - - - 4 15 28 17 88 31 10]
59 [22 88 - - 4 15 28 17 59 31 10]

final table: [22 88 - - 4 15 28 17 59 31 10]

Double Probing using $h_1(x) = x\ mod\ 11$ and $h_2(x) = 7$ - ($x\ mod\ 7$):

10 [- - - - - - - - - - 10]
22 [22 - - - - - - - - - 10]
31 [22 - - - - - - - 31 10]
4 [22 - - - 4 - - - - 31 10]
15 [22 - - - 4 15 - - - 31 10]
28 [22 - - - 4 15 28 - - 31 10]
17 [22 - - 17 4 15 28 - - 31 10]
88 [22 - - 17 4 15 28 - 88 31 10]
59 [22 88 - 17 4 15 28 - 59 31 10]

final table: [22 88 - 17 4 15 28 - 59 31 10]

Exercise 2:

For the array with elements: 3 1 2 4 5 8 7 6 9

The elements that could have been the pivot element after the Quicksort are 4,5, and 9, since they each have smaller elements to the left of them in the array and larger elements to the right of them in the array.

Exercise 3.

Answer: $1 - 2\alpha$.

When the partition function produces a split where each subproblem yields $\alpha * n$ elements, the pivot must fall between the indices $\alpha * n$, $(1 - \alpha) * n$. The length of this range would be $(1 - \alpha) * n - \alpha * n = 1 - 2\alpha * n$. Dividing the total range, n, from this valid range would yield $1 - 2\alpha$.

Programming Task 6

Test data 1 Preorder:
(7,6) (3,1) (10,4) (9,1) (13,2) (11,1)

Test Data 1 leftRotate: ✓
(10,6) (7,3) (3,1) (9,1) (13,2) (11,1)

input-6-1.txt Preorder:
(448,1000) (184,447) (43,187) (10,43) (4,8) (0,4) (3,3) (1,1) (4,1) (9,3) (5,2) (8,1) (32,34) (23,23) (11,12) (13,11) (12,2) (12,1) (16,8) (14,2) (15,1) (23,5) (21,4) (18,2) (20,1)

input-6-1.txt leftRotate: ✓
(964,1000) (448,973) (184,447) (43,187) (10,43) (4,8) (0,4) (3,3) (1,1) (4,1) (9,3) (5,2) (8,1) (32,34) (23,23) (11,12) (13,11) (12,2) (12,1) (16,8) (14,2) (15,1) (23,5) (21,4) (18,2)

input-6-2.txt Preorder:
(745,10000) (151,767) (8,141) (3,6) (2,2) (3,1) (6,3) (4,2) (4,1) (105,134) (63,86) (63,48) (9,47) (54,46) (21,38) (21,10) (20,9) (18,8) (18,7) (16,6) (14,4) (11,2) (12,1) (16,1) (18,1)

✓

input-6-2.txt leftRotate:
(5102,10000) (745,5096) (151,767) (8,141) (3,6) (2,2) (3,1) (6,3) (4,2) (4,1) (105,134) (63,86) (63,48) (9,47) (54,46) (21,38) (21,10) (20,9) (18,8) (18,7) (16,6) (14,4) (11,2) (12,1) (16,1)

```java
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;

/** Since only 2 files are submitted (pdf and .java) all classes are stuffed into this file */
public class Assignment6 {
    public static void main(String[] args) {
        Node test_data1 = insert(7, 10, 3, 9, 13, 11); //test data 1
        Node input6_1 = inputFile("input-6-1.txt");
        Node input6_2 = inputFile("input-6-2.txt");

        /*Test data 1: insert 7, 10, 3, 9, 13, 11.
        Your program will print: (7,6), (3,1), (10,4),(9,1), (13, 2), (11,1).
        Next, do a leftRotate, and print the tree after rotation and you get (10,6), (7,3), (3,1),
(9,1) (13,2), (11, 1). */
        System.err.println("Test data 1:");
        preorder(test_data1);
            System.err.println("\n\nTest data 1 After leftRotation");
            preorder(leftRotate(test_data1));


        //first 25 of 6-1: (448,1000) (184,447) (43,187) (10,43) (4,8) (0,4) (3,3) (1,1) (4,1) (9,3)
(5,2) (8,1) (32,34) (23,23) (11,12) (13,11) (12,2) (12,1) (16,8) (14,2) (15,1) (23,5) (21,4) (18,2)
(20,1)
        System.err.println("\n\nPreorder traversal of input6_1:");
        preorder(input6_1);

        System.err.println("\n\nLeft Rotation of input6_1:");
        preorder(leftRotate(input6_1));

        //first 25 of 6-2: (745,10000) (151,767) (8,141) (3,6) (2,2) (3,1) (6,3) (4,2) (4,1)
(105,134) (63,86) (63,48) (9,47) (54,46) (21,38) (21,10) (20,9) (18,8) (18,7) (16,6) (14,4) (11,2)
(12,1) (16,1) (18,1)
        System.err.println("\n\nPreorder traversal of input6_2:");
        preorder(input6_2);

        System.err.println("\n\nLeft Rotation of input6_2:");
        preorder(leftRotate(input6_2));
    }

    /** reads the file and converts it to an Node with children/binarytree */
    public static Node inputFile(String filename) {
        try (Scanner console = new Scanner(new FileReader(filename))) {


            final int N = console.nextInt();
            Node root = new Node(console.nextInt());

            //reads ints in the file ; starts at 1 because root is already read
            for (int i = 1; i < N; i++)
                insert(root,console.nextInt());

            return root;
        }catch(FileNotFoundException e) {
            System.err.println("File not found: '" + filename + "'");
        }

        return null;
    }
    /** function to search a key in a BST.
     * @param root
     * @param KEY
     * @return the {@code node} with key or {@code null} if doesnt exist
     */
    static Node search(Node root, final int KEY)
    {
        // Base Cases: root is null or key is already the root
        if (root == null || root.key == KEY)
            return root;

        // Key is greater than root's key
        if (KEY > root.key)
            return search(root.right, KEY);

        // Key is smaller than root's key
        return search(root.left, KEY);
    }


    /** rotates the root t to the left, so that the right child of {@code t} becomes the parent of t,
and symmetrically rightRotate  (Node t)
     * @param t the node that gets rotated left
    */
    static Node leftRotate(final Node t) {
        if (t == null || t.right == null) return t; // no right child to rotate

        final Node L_rotate = t.right;
            L_rotate.size = t.size; // size of left rotate = old root;

        t.right = L_rotate.left;
            t.size = 1;
            if (t.right != null) t.size += t.right.size; // add size of right child
            if (t.left != null) t.size += t.left.size; // add size of left child

        L_rotate.left = t;

        return L_rotate;
    }
```

ok!

```
    /** opposite of leftRotate
     * @param t the node that gets rotated right
    */
    static Node rightRotate(final Node t){
        if (t == null || t.left == null) return t; // no left child to rotate


        final Node R_rotate = t.left;
            R_rotate.size = t.size; // size of right rotate = old root;

        t.left = R_rotate.right;
            t.size = 1;
            if (t.right != null) t.size += t.right.size; // add size of right child
            if (t.left != null) t.size += t.left.size; // add size of left child

        R_rotate.right = t;

        return R_rotate;
    }

    /** function to insert a key in a BST
     * @param root
     * @param KEY
     * @return root node with inserted KEY
    */
    static Node insert(Node root, final int KEY) {
        if (root == null) return new Node(KEY);

        if (KEY <= root.key) root.left = insert(root.left, KEY); //duplicates go to left
        else if (KEY > root.key) root.right = insert(root.right, KEY);
        root.size++;

        return root;
    }

    /** exists for easier testing
     * @param KEYS
     * @return new node with a root of the first element of {@code KEYS}
     * <br> exists for easier testing
     */
    static Node insert(final int... KEYS){
        Node root = new Node(KEYS[0]);

        for (int i = 1; i < KEYS.length; i++)
            insert(root, KEYS[i]);

        return root;
    }

    /** prints preorder (key,size) */
    static void preorder(Node root){
        if (root == null) return;

        System.out.printf(
            "(%d,%d) ",
            root.key, root.size
        );

        preorder(root.left);
        preorder(root.right);
    }


}

    class Node {
        /**  keeps the number of nodes
     in the tree rooted at that node (including in the count the node itself). The constructors
    and the insertion function need to take into account the sizes of the nodes. */
        public int size = 0;

        public final int key;

        public Node left, right;

        public Node(int item)
        {
            key = item;
            left = right = null;

            size = 1;
        }


    }
```