Algorithms and Data Structures Lab 5 Preparation

Authors: Karim Haidar
Charithma Perera

June 12, 2023

Contents

1 Problem 13 (Common Preparation)

1.1 Part1

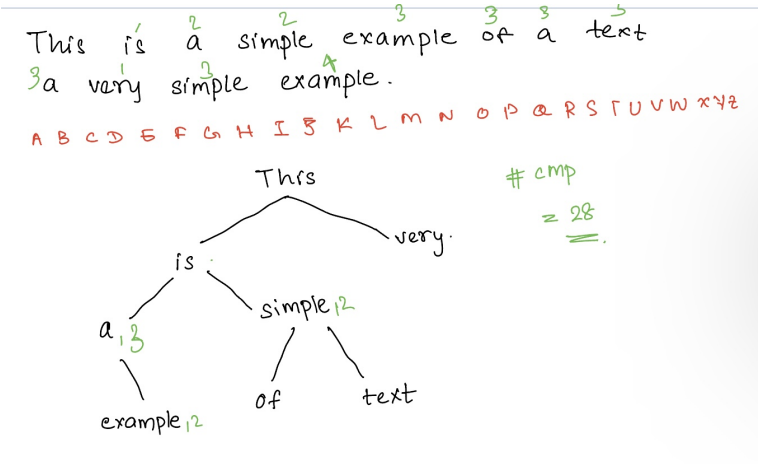

Figure 1: Binary search tree for the text

Word	No of comparisons	Node depth
This	0	0
is	1	1
a	2	2
simple	2	2
example	3	3
of	3	3
a	3	2
text	3	3
a	3	2
very	1	1
simple	3	2
example	4	3

- Total no of comparisons = 28

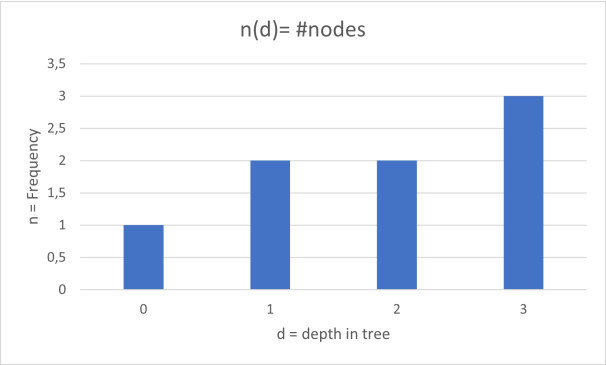

Figure 2: Histogram for node depths [4]

- Tree depth = 3
- Final average = 1.875 (from the excel sheet)
- standard deviation = 1.05 (from excel sheet)

1.2 Part 2

- To refine the Node class to store the depth of a node in the tree, an additional field should be added to keep track of the depth.

```
        private class Node {
    private Key key;
    private Value value;
    private Node left, right;
    private int size;
    private int depth;

    public Node(Key key, Value value, int depth) {
        this.key = key;
        this.value = value;
        this.size = 1;
        this.depth = depth;
    }
}
```

- Modification of put method

```
        private Node put(Node x, Key key, Value val, int depth) {
    if (x == null)
        return new Node(key, val, depth);

    int cmp = key.compareTo(x.key);
    if (cmp == 0)
        x.value = val;
    else if (cmp < 0)
        x.left = put(x.left, key, val, depth + 1);
    else if (cmp > 0)
        x.right = put(x.right, key, val, depth + 1);

    x.size = 1 + size(x.left) + size(x.right);
    return x;
}

@Override
public void put(Key key, Value value) {
    if (key == null)
        throw new IllegalArgumentException("Key cannot be null.");
    root = put(root, key, value, 0);
}
```

- Code to calculate the depth of the node

```
      private int calculateDepth(Node node, Key key) {
    if (node == null)
        return -1; // Key not found in the BST

    int cmp = key.compareTo(node.key);
    if (cmp == 0)
        return 0; // Found the node, so depth is 0
    else if (cmp < 0) {
        int leftDepth = calculateDepth(node.left, key);
        if (leftDepth != -1)
            return 1 + leftDepth;
    } else {
        int rightDepth = calculateDepth(node.right, key);
        if (rightDepth != -1)
            return 1 + rightDepth;
    }
```

```
        return -1; // Key not found in the subtree
    }

}
```

To retrive the histogram we can iterate over the keys in the BST and calculate the depth of each node by calling the calculateDepth method for each key. and by inserting these values to excel table we can obtain a histogram,

```
    BST<Key, Value> bst = new BST<>();
// Perform insertions and build the BST

for (Key key : bst.keys()) {
    int depth = calculateDepth(bst.root, key);
    System.out.println("Key: " + key + ", Depth: " + depth);
}
```

1.3 Part 3

The probability p of calling the `putRoot` function at each insert call is $\frac{1}{N+1}$, where N represents the current size of the node x.

This means that as the tree grows, the probability of calling the `putRoot` function decreases. When the tree is empty ($N = 0$), the probability is 1, and as more nodes are added, the probability decreases.

Insert Call	Probability (p)
0	1
1	0.5
2	0.3
3	0.25
4	0.2
5	0.16
6	0.14
7	0.125

1.4 Part 4

Common Properties:

1. Key-Value Pairing: Both ADTs associate a value with a unique key.
2. Insertion and Retrieval: Both ADTs support operations to insert key-value pairs (`put`) and retrieve the value associated with a key (`get`).
3. Key Existence Check: Both ADTs provide a method to check if a key exists in the table (`contains`).

Differences:

1. Type Constraints: The `ST<Key, Value>` class uses a type constraint (`Key extends Comparable<Key>`) to ensure that the keys are comparable. In contrast, the `java.util.Map` interface does not impose such a constraint.
2. Implementation: The `java.util.Map` interface is an interface that defines the functionality, while the `ST<Key, Value>` class is an implementation of a symbol table.
3. Java Collection Framework: The `java.util.Map` interface is part of the Java Collection Framework, providing a standardized way to work with key-value mappings. On the other hand, the `ST<Key, Value>` class is a custom implementation.

Regarding the hint about `TreeSet`:

1. `TreeSet` throws a `ClassCastException` when the objects being stored are not mutually comparable (i.e., the elements don't implement the `Comparable` interface).
2. The concept that makes the difference in the two ADT definitions is the ability to compare elements. In `java.util.TreeSet`, the elements must be mutually comparable, allowing them to be stored in a sorted order based on their natural ordering or a custom comparator.

2 Lab

2.1 Problem 14

Implemented a BST-Iterator and tested it by printing it into a .txt file. (Took assistance from Rushit and Ruben St.Martin) [6]

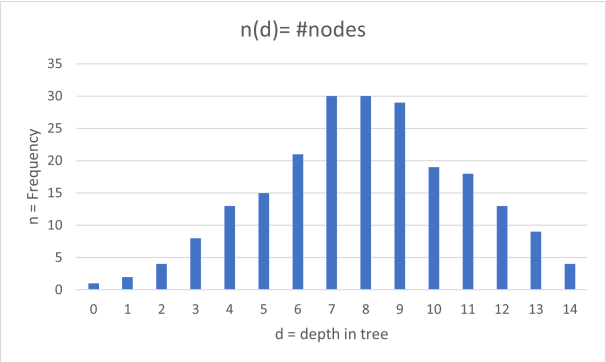

Figure 3: Histogram plotted for the long text

Excel sheet is attached in the file.

2.2 Problem 15

The objective of this problem is to investigate and determine the average depth of a binary search tree (BST) as a function of the number of nodes inserted. The depth of the tree, which refers to the maximum node depth, is influenced by the order in which items are input into the tree. In the worst-case scenario, the depth grows linearly with the number of nodes N. However, we aim to determine the growth pattern in the average case, such as N1/2 or log(N). For this, random sequences of Doubles are generated to simulate different BSTs. Considered M=20 sequences with exponentially growing N values (N=2∧ k - 1 for k=3, 4, 5, 6, ..., kmax). The "treeDepth" Method is Implemented to calculate BST depth by traversing the tree arbitrarily and finding the maximum node depth. Tested with N=7 Doubles. Then the data is plotted using double-log and semi-log graphs to validate N1/2 and log(N) hypotheses.

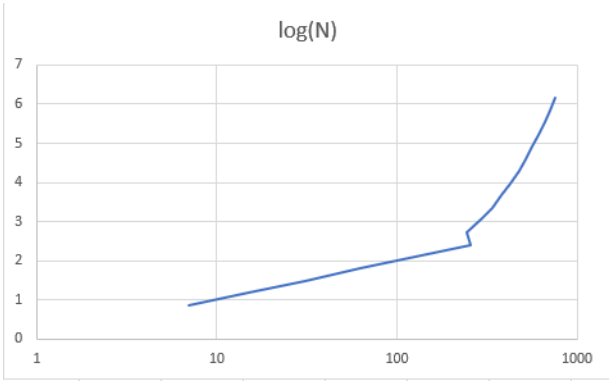

Figure 4: semi-log plot to check the log(N) hypothesis

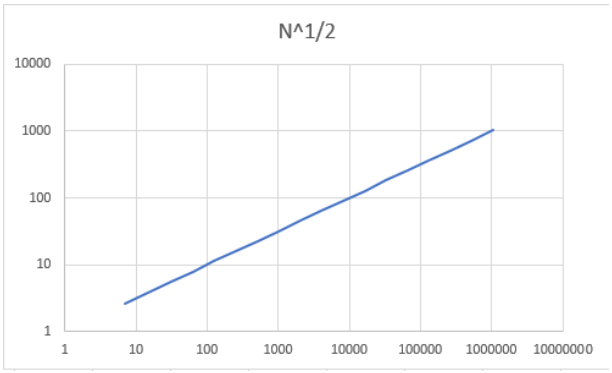

Figure 5: double-log plot to check the N1/2 hypothesis

OPERATION	WORST CASE	AVERAGE CASE	BEST CASE	SPACE
Search	O(N)	O(logN)	O(1)	O(N)
Insert	O(N)	O(logN)	O(1)	O(N)
Delete	O(N)	O(logN)	O(N)	O(N)

Figure 6: [7]

We can then conclude that the log(N) hypothesis is true based on the comparison between the plots and the theory.

References

[1] Sedgewick Slides. (n.d.). In R. Sedgewick and K. Wayne, Algorithms, 4th Edition. Addison-Wesley Professional.

[2] StackOfDoubles class. (n.d.). In Princeton University, Algorithms, Part I. Retrieved from https://algs4.cs.princeton.edu/13stacks/StackOfDoubles.java.html

[3] Stack overflow https://stackoverflow.com/

[4] Algorithms-Datastructures IE3 (Renz) SoSe 2023 https://e-assessment.haw-hamburg.de/course/view.php?id=391#section-0

[5] https://www.geeksforgeeks.org/

[6] Lab report 5- Rushith and Ruben St.Martin

[7] OpenGenus IQ https://iq.opengenus.org/