tim allec

Name, SID, Date ....

In Class Assignment 22: Optimal Binary Search Tree Data Structure

Benjamin Sanders, MS November 25, 2020

1 Introduction

You may work in groups of up to two or three students. Write all student names at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

import java.util.Vector;

....
....
....
....

2 Assignment Description

2.1 Big Picture

This Dynamic Programming algorithm is used to construct Binary Search Trees that minimize expected search cost e.

2.2 Algorithm Implementation

Implement the following algorithm in Java, using the Vector data structure for any 1-D array, 2-D array (a.k.a. table), or linear algebra purposes.

Name, SID, Date ....

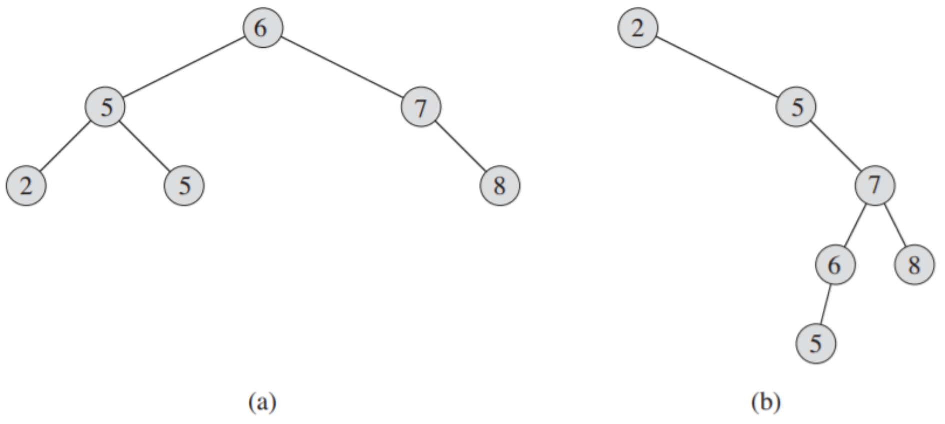

Figure 12.1 Binary search trees. For any node x, the keys in the left subtree of x are at most $x.key$, and the keys in the right subtree of x are at least $x.key$. Different binary search trees can represent the same set of values. The worst-case running time for most search-tree operations is proportional to the height of the tree. **(a)** A binary search tree on 6 nodes with height 2. **(b)** A less efficient binary search tree with height 4 that contains the same keys.

The keys in a binary search tree are always stored in such a way as to satisfy the ***binary-search-tree property***:

> Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $y.key \leq x.key$. If y is a node in the right subtree of x, then $y.key \geq x.key$.

Thus, in Figure 12.1(a), the key of the root is 6, the keys 2, 5, and 5 in its left subtree are no larger than 6, and the keys 7 and 8 in its right subtree are no smaller than 6. The same property holds for every node in the tree. For example, the key 5 in the root's left child is no smaller than the key 2 in that node's left subtree and no larger than the key 5 in the right subtree.

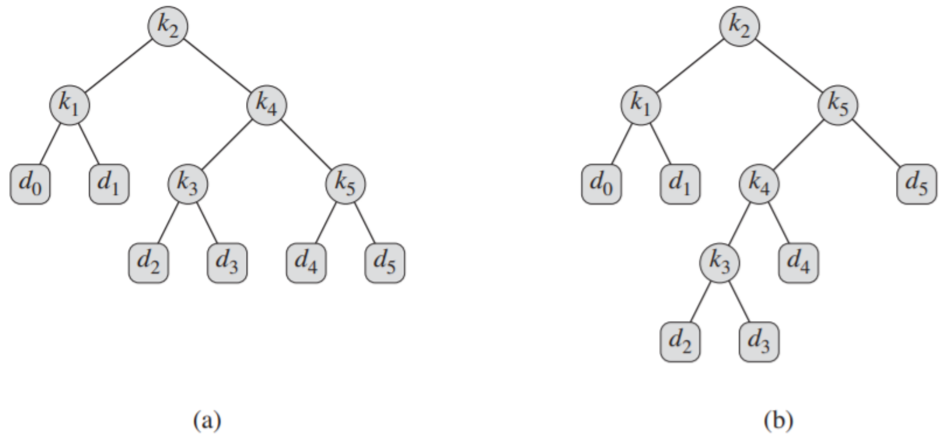

Figure 15.9 Two binary search trees for a set of $n = 5$ keys with the following probabilities:

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

(a) A binary search tree with expected search cost 2.80. **(b)** A binary search tree with expected search cost 2.75. This tree is optimal.

Name, SID, Date ....

Note that d_i refers to a “dummy” or NULL node.

```
OPTIMAL-BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
for i = 1 to n + 1
    e[i, i - 1] = q_{i-1}
    w[i, i - 1] = q_{i-1}
for l = 1 to n
    for i = 1 to n - l + 1
        j = i + l - 1
        e[i, j] = ∞
        w[i, j] = w[i, j - 1] + p_j + q_j
        for r = i to j
            t = e[i, r - 1] + e[r + 1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] = t
                root[i, j] = r
return e and root
```

Note that you should specify the values for p and q, based on how frequently you expect a given key to appear in a context. p and q range between 0 and 1, and should be considered probability indicators for each key. For example, if you consider the 26 letters of the English alphabet, you might recognize that E, A, R, I, O, T, N, and S are the most commonly used letters. Therefore, you could choose to assign $p_0 = .83$, $q_0 = .63$ for A because it is an often-used letter. The exact choice of p and q is specific to each application, and we therefore won’t elaborate on that here. Therefore, you should arbitrarily choose p and q values between 0 and 1 for your implementation. In the OPTIMAL-BST algorithm, note that p and q are arrays of these decimal values, each array being of size n.

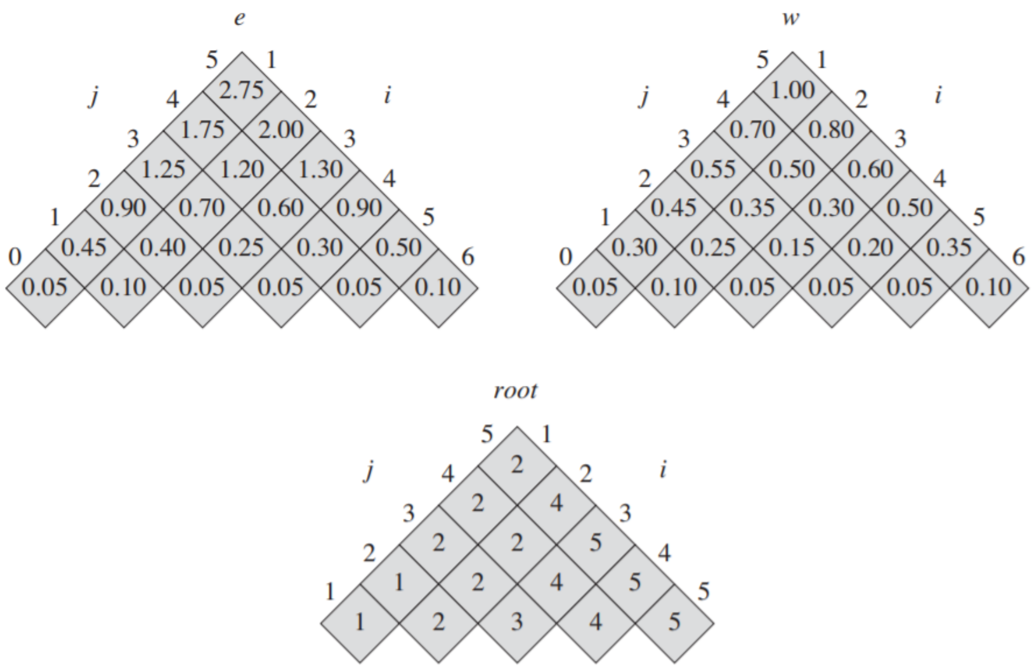
e
j
i
2.75
1.75 2.00
1.25 1.20 1.30
0.90 0.70 0.60 0.90
0.45 0.40 0.25 0.30 0.50
0.05 0.10 0.05 0.05 0.05 0.10
w
1.00
0.70 0.80
0.55 0.50 0.60
0.45 0.35 0.30 0.50
0.30 0.25 0.15 0.20 0.35
0.05 0.10 0.05 0.05 0.05 0.10
root
2
2 4
2 2 5
1 2 4 5
1 2 3 4 5

Figure 15.10 The tables $e[i, j]$, $w[i, j]$, and $root[i, j]$ computed by OPTIMAL-BST on the key distribution shown in Figure 15.9. The tables are rotated so that the diagonals run horizontally.

Name, SID, Date ....

2.3 Time Complexity Analysis

Where n is the number of data points to be stored, analyze the time complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n^2)$ in the space below.

o(log n)

2.4 Space Complexity Analysis

Where n is the number of data points to be stored, analyze the space complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n \cdot log(n))$ in the space below.

O(n^2)

3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.