Binary Search Tree with Strings

Last updated: June 8, 2023

Written by: baeldung (https://www.baeldung.com/cs/author/baeldung)

Algorithms (https://www.baeldung.com/cs/category/algorithms)

Trees (https://www.baeldung.com/cs/category/graph-theory/trees)

Binary Tree (https://www.baeldung.com/cs/tag/binary-tree)

Strings (https://www.baeldung.com/cs/tag/strings)

1. Overview

In this tutorial, we'll talk about the **binary search tree (BST)** data structure focusing more on the case **where the keys in the nodes are represented by strings.**

2. Introduction

A BST (/cs/binary-search-trees) is a binary tree (https://en.wikipedia.org/wiki/Binary_tree) that satisfies the following properties for every node:

- The left subtree of the node contains only nodes with keys lesser than the node's key
- The right subtree of the node contains only nodes with keys greater than the node's key
- The left and right subtree each must also be a binary search tree.

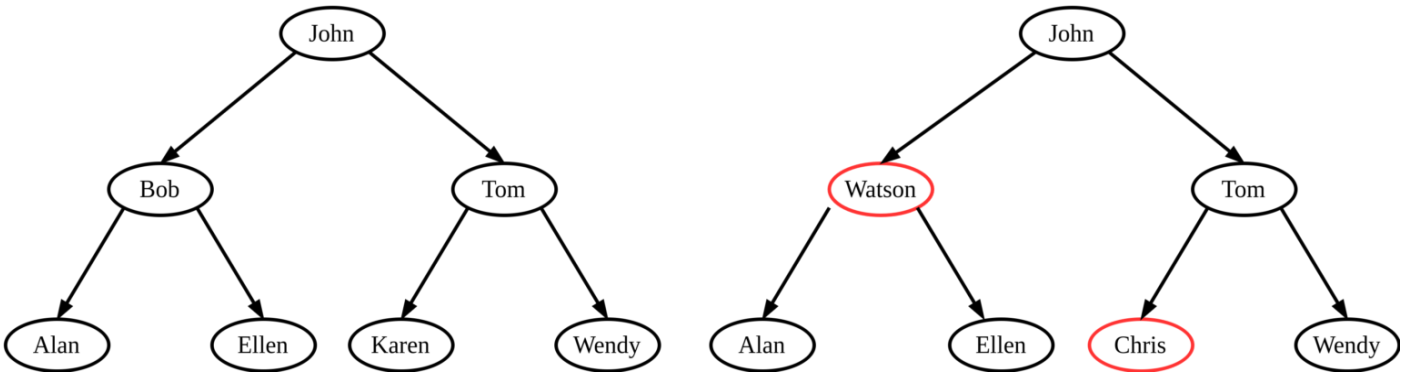
In this article, **we focus more on the case where the keys of the nodes are represented by strings and not numbers.** In this case, we should first define the ordering of the strings.

Lexicographic ordering is defined as the order that each string that appears in a dictionary. To determine which string is lexicographically larger we compare the corresponding characters of the two strings from left to right. The first character where the two strings differ determines which string comes first. Characters are compared using the Unicode (/java-char-encoding) character set and all uppercase letters come before lowercase letters.

For example:

- apple < orange because ‘a’ comes before ‘o’
- Orange > orange because ‘O’ comes after ‘o’
- apple = apple because all letters are the same

Let’s take a look at two binary trees with strings:



The left tree is a BST since it satisfies the above criterion while the right tree is not a BST because in the red nodes the criterion fails. ‘Watson’ is lexicographically larger than ‘John’ and ‘Chris’ is lexicographically smaller than ‘John’.

3. Basic Operations

As the name suggests, the most frequent operation in a BST with strings is searching for a specific string. Starting from the root we follow a downward path until we find the requested string.

For each node we encounter, we lexicographically compare the requested string S_r with the node’s string S_n. If $S_r < S_n$, we continue the search in the left subtree since the BST property implies that S_r could not be stored in the right subtree. Symmetrically, if $S_r > S_n$ we search the right subtree. The whole procedure is summarized in the above pseudocode:

Algorithm 1: Search for a string S in a BST

```
Data: A BST and a query string S
Function BST-Search(node, S):
    if node == NULL then
        return 'S not found'
    end
    if S = node.key then
        return node;
    end
    if S < node.key then
        return BST-Search(node.left, S);
    end
    return BST-Search(S, node.right);
```

Other useful operations are to insert or delete a specific string from the BST. The data structure must be modified to reflect this change, but in such a way that the binary-search-tree property continues to hold. When inserting a new string (/cs/inserting-complexity-binary-search-tree), we basically search for the string in the tree using the *BST-Search* method. When we reach the NULL pointer (since the string is not present in the tree), we insert a node that contains the input string:

Algorithm 2: Insert a node z in a BST

```
Data: A BST and a node z to insert
Function BST-Insert(T, z):
    y = NULL;
    x = T.root;
    while x != NULL do
        y = x;
        if z.key < x.key then
            x = x.left;
        else
            x = x.right;
        end
    end
    z.p = y;
    if y == NULL then
        T.root = z;
    else if z.key < y.key then
        y.left = z;
    else
        y.right = z;
    end
```

The process of deletion is slightly more intricate. There are three cases:

- If the node has no children, we simply remove it
- If the node has just one child, we replace the node with its child
- If the node has two children, we find its successor that lies in the right subtree and has no left child. Then, we replace the node with its successor

Let's see the pseudocode:

Algorithm 3: Delete a node z in a BST

```
Data: A BST and a node z to delete
Function BST-Delete(T, z):
    if z.left == NULL then
        TRANSPLANT(T, z, z.right);
    else if z.right = NULL then
        TRANSPLANT(T, z, z.left);
    else
        y = TREE-MINIMUM(z.right);
        if y.p != z then
            TRANSPLANT(Y, y, y.right);
            y.right = z.right;
            y.right.p = y;
        end
        TRANSPLANT(T, z, y);
        y.left = z.left;
        y.left.p = y;
    end
```

In order to move subtrees within the binary search tree, we define a subroutine TRANSPLANT, which replaces one subtree as a child of its parent with another subtree. When TRANSPLANT replaces the subtree rooted at node a with the subtree rooted at node b, node a's parent becomes node b's parent, and a's parent ends up having b as its appropriate child.

The pseudocode for the TRANSPLANT function is as follows:

Algorithm 4: Move subtrees around within the binary search tree

```
Data: A BST and two nodes a and b
Function TRANSPLANT(T, a, b):
    if a.p == NULL then
        T.root=b;
    else if a == a.p.left then
        a.p.left = b;
    else
        a.p.right = b;
    end
    if b ≠ NULL then
        b.p = a.p
    end
```

4. Complexity Analysis

In a BST searching, insertion and deletion run in time $O(h)$ time where h is the height of the tree:

Operation	Time Complexity
Search	$\mathcal{O}(h)$
Insertion	$\mathcal{O}(h)$
Deletion	$\mathcal{O}(h)$

5. Conclusion

In this article, we presented the basic operations of a BST that contains strings as keys.

Comments are closed on this article!

CATEGORIES

ALGORITHMS (/CS/CATEGORY/ALGORITHMS)
ARTIFICIAL INTELLIGENCE (/CS/CATEGORY/AI)
CORE CONCEPTS (/CS/CATEGORY/CORE-CONCEPTS)
DATA STRUCTURES (/CS/CATEGORY/DATA-STRUCTURES)
GRAPH THEORY (/CS/CATEGORY/GRAPH-THEORY)
LATEX (/CS/CATEGORY/LATEX)
NETWORKING (/CS/CATEGORY/NETWORKING)
SECURITY (/CS/CATEGORY/SECURITY)

SERIES

ABOUT

ABOUT BAELDUNG (HTTPS://WWW.BAELDUNG.COM/ABOUT)
THE FULL ARCHIVE (/CS/FULL_ARCHIVE)
EDITORS (HTTPS://WWW.BAELDUNG.COM/EDITORS)

TERMS OF SERVICE (HTTPS://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTPS://WWW.BAELDUNG.COM/PRIVACY-POLICY)
COMPANY INFO (HTTPS://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)