Miguel Alcón Doganoc
Advanced Data Structures
June 21, 2019

Dictionary with Binary Search Trees

Introduction

The objective of this work is to implement a dictionary using a binary search tree (BST) as the base of the data structure, and then compare it with the default *dict* class of python. I started my implementation from the *Python* BST implementation in [1].

Implementation

First, I moved the code from *Python 2* to *Python 3* because I am used to work with *Python 3*. After that, I modified the *Node* class to let it accept a *key* and a *value*. The main idea of this modification is to use the *key* as the normal value of a BST, and add a new parameter in the node saving the *value* corresponding to the *key* of the node. In my opinion, this is a simple modification that adds a great functionality to the data structure.

Regarding the operations, all the standard ones were done in the implementation that I took from the Internet, but the deletion operation was not completed, and the insertion operation does not behave as I wanted. The deletion operation did not take into account that a node can have no parent, hence when I wanted to delete the root node of the main tree, it did nothing. For each of the cases (0, 1 and more than 1 children), I implemented the behavior of the deletion operation for the root node of the main tree. For the insertion operation, I had to force the tree to not allow multiple *keys* by modifying the operation itself. Instead of adding a pair *key-value* without checking if the *key* is already in the tree, now the BST dictionary checks it and swaps the previous *value* with the new one, as the *Python* dictionary behaves.

Experiment

As it is known, the *Python* dictionary allows the *key* to be whatever hashable class, and the *value* whatever class. In my case, the *key* has to be a comparable and ordered class. With my own implementation I tried to insert different types of *values* (boolean, dictionaries and lists), with different types of *keys* (integer, float, list and dictionaries).

After that, I tested the performance of the main operations of the BST dictionary and the *Python* one. In order to do that, I generated randomly 1000 integer *keys* and another 1000 integer *values*. First, I inserted them to both dictionaries, then I searched each of the keys, and finally I deleted all of them one by one.

Results

In the type acceptance test, only when I tried to insert a pair *key-value* with a dictionary as a *key*, the BST crashed. I checked it for the *Python* one and it also failed.

In the performance test, the *Python* dictionary outperforms the BST one, as you can see in table 1.

Operation	Python	BST
Insertion	0.3516	28.728
Search	0.2737	4.059
Deletion	0.3114	3.905

Table 1: Performance results (in milliseconds)

References

[1] NahimNasser (GitHub). Binary tree. https://gist.github.com/NahimNasser/4705371, 2012.