Advanced Data Structures

Amalia Duch

CS-UPC

February 19, 2020

Definition

A binary search tree T is a binary tree that either it is empty or it's root contains an element x satisfying that

1. the left and right subtrees of T, L and R, are, respectively, binary search trees.
2. For every element y of L, $y < x$, and for every element z of R, $z > x$.

Lemma

An inorder traversal of a binary search tree T visits all the elements of T by increasing order of its keys.

A BST of n elements can be equivalent to a list since it can contain a node with no children and $n-1$ nodes with exactly one child (if we insert, for instance, a sequence of n elements with keys in ascending order in an initially empty BST). Such a BST has height n. In the worst case, the cost of a search, an insertion or a deletion is $\Theta(n)$. In general, a search, insertion or deletion in a BST of height h will will have worst case cost $\Theta(h)$. As a function of n, the height can arrive to a value of n, therefore, the worst case cost of the search and update operations is $\Theta(n)$.

But in the average, the cost of these operations will be smaller. Let us suppose that any order of insertion of the elements in the BST is equiprobable. For successful search, let us suppose that we look for any of the n keys with equal probability of $1/n$. For unsuccessful search or insertions, let us suppose that we arrive to any of the $n+1$ leaves with equal probability.
The cost of any of these operations will be proportional to the number of required comparisons between the given key and the keys of the examined nodes. Let $C(n)$ be the average number of comparisons and $C(n;k)$ the average number of comparisons when the root is occupied by the k-th key

$$C(n) = \sum_{1 \leq k \leq n} C(n;k) \times \mathbb{P}[\text{root is the } k\text{-th}] \,.$$

$$
\begin{aligned}
C(n) &= \frac{1}{n} \sum_{1 \le k \le n} C(n;k) \\
&= 1 + \frac{1}{n} \sum_{1 \le k \le n} \left(\frac{1}{n} \cdot 0 + \frac{k-1}{n} \cdot C(k-1) + \frac{n-k}{n} \cdot C(n-k) \right) \\
&= 1 + \frac{1}{n^2} \sum_{0 \le k < n} \left(k \cdot C(k) + (n-1-k) \cdot C(n-1-k) \right) \\
&= 1 + \frac{2}{n^2} \sum_{0 \le k < n} k \cdot C(k).
\end{aligned}
$$

Another way to see it is to calculate $I(n)$, the average internal path length (IPL), which in a BST is the sum of the distances from the root to every node in the tree.

$$C(n) = 1 + \frac{I(n)}{n}$$

The average IPL satisfies the recurrence

$$I(n) = n - 1 + \frac{2}{n} \sum_{0 \le k < n} I(k), \qquad I(0) = 0. \tag{1}$$

This is because every node except the root contributes to the IPL with 1 plus the contribution to the IPL of the subtree beneath it.

In order to solve the recurrence of $I(n)$ we calculate $(n+1)I(n+1) - nI(n)$:

$$\begin{aligned}(n+1)I(n+1) - nI(n) &= (n+1)n - n(n-1) + 2I(n) \\ &= 2n + 2I(n); \\ (n+1)I(n+1) &= 2n + (n+2)I(n)\end{aligned}$$

$$
\begin{aligned}
I(n+1) &= \frac{2n}{n+1} + \frac{n+2}{n+1} I(n) = \frac{2n}{n+1} + \frac{2(n-1)(n+2)}{n(n+1)} + \frac{n+2}{n} I(n-1) \\
&= \frac{2n}{n+1} + \frac{2(n-1)(n+2)}{n(n+1)} + \frac{2(n-2)(n+2)}{n(n-1)} + \frac{n+2}{n-1} I(n-2) \\
&= \frac{2n}{n+1} + 2(n+2) \sum_{i=1}^{i=k} \frac{n-i}{(n-i+1)(n-i+2)} + \frac{n+2}{n-k+1} I(n-k) \\
&= \frac{2n}{n+1} + 2(n+2) \sum_{i=1}^{i=n} \frac{i}{(i+1)(i+2)} \\
&= \mathcal{O}(1) + 2(n+2) \sum_{1 \le i \le n} \left(\frac{2}{i+2} - \frac{1}{i+1} \right) \\
&= \mathcal{O}(1) + 2(n+2)(\frac{2}{n+2} + \frac{1}{n+1} + H_n - 2) \\
&= 2nH_n - 4n + 4H_n + \mathcal{O}(1),
\end{aligned}
$$

where $H_n = \sum_{1 \leq i \leq n} 1/i$.
Since $H_n = \ln n + \mathcal{O}(1)$ it follows that

$$\begin{aligned} I(n) = Q(n) &= 2n \ln n + \mathcal{O}(n) \\ &= 1.386 \ldots n \log_2 n + \mathcal{O}(n) \end{aligned}$$

and that $C(n) = 2 \ln n + \mathcal{O}(1)$.


J. L. Bentley.
Decomposable searching problems.
Information Processing Letters, 8(5):133–136, 1979.


J. L. Bentley and J. H. Friedman.
Data structures for range searching.
ACM Computing surveys, 11(4):397–409, 1979.


J.L. Bentley and J. H. Friedman.
Data structures for range searching.
ACM Computing Surveys, 11(4):397–409, 1979.


W. A. Burkhard.
Hashing and trie algorithms for partial match retrieval.
ACM Transactions on Database Systems, 1(2):175–187, 1976.


L. Devroye.

Branching processes in the analysis of the height of trees.
Acta Informatica, 24:277–298, 1987.


L. Devroye and L. Laforest.
An analysis of random d-dimensional quadtrees.
SIAM Journal on Computing, 19(5):821–832, 1990.


R. A. Finkel and J. L. Bentley.
Quad trees: a data structure for retrieval on composite key.
Acta Informatica, 4(1):1–9, 1974.


Ph. Flajolet, G. Gonnet, C. Puech, and J. M. Robson.
Analytic variations on quad trees.
Algorithmica, 10:473–500, 1993.


Ph. Flajolet and T. Lafforgue.
Search costs in quad trees and singularity perturbation analysis.

Discrete and Computational Geometry, 12(4):151–175, 1993.


M. Friedman, F. Baskett, and L. J. Shustek.
An algorithm for finding nearest neighbors.
IEEE Transactions on Computing, C–24(10):1000–1006, 1975.


D. E. Knuth.
The Art of Computer Programming: Sorting and Searching, volume 3.
Addison–Wesley, 2nd edition, 1998.


J. Nievergelt, H. Hinterberger, and K. C. Sevcik.
The grid file: An adaptable symmetric multikey file structure.
ACM Transactions on Database Systems, 1(9):38–71, 1984.


M. Regnier.
Analysis of the grid file algorithms.
BIT, 25(2):335–357, 1985.


R. L. Rivest.
Partial-match retrieval algorithms.
SIAM Journal on Computing, 5(1):19–50, 1976.


H. Samet.
Deletion in two-dimensional quad-trees.
Communications of the ACM, 23(12):703–710, 1980.


H. Samet.
The Design and Analysis of Spatial Data Structures.
Addison-Wesley, 1990.