Improving Height-Balance in Search Trees: center versus root, radius versus height

KOBA GELASHVILI
Department of Computer Sciences
I. Javakhishvili Tbilisi State University
2, University st., 0143, Tbilisi
GEORGIA
kobage@gmail.com

IRINA KHUTSISHVILI
Department of Computer Sciences
I. Javakhishvili Tbilisi State University
2, University st., 0143, Tbilisi
GEORGIA
i.khutsishvili@yahoo.com

Abstract: - In Order to hasten basic operations on some popular data structures of implementation height-balanced BSTs, applying “center-radius” technique instead of “root-height” is suggested. Reasonability of suggested approach on concrete samples of BSTs, taken from real applications, as well as possible complications in program codes are considered.

Key-Words: - BST, RBT, AVL, root, height, center, radius, height-balanced.

1 Motivation and scientific approach

In computer science, a height-balanced binary search tree is a binary search tree (BST) that attempts to keep its height, or the number of levels of nodes beneath the root, as small as possible at all times, dynamically.

Most operations on a binary search tree take time directly proportional to the height of the tree, so it is desirable to keep the height small. Some of popular data structures of implementing height-balanced binary search trees include:

- AVL tree;
- Red-black tree;
- Splay tree.

Times for several basic functions are shown at the table below, where n is a number of nodes in the tree. For some implementations these times are worst-case, while for others they are amortized (see [1], [2], [3]).

<table>
<thead>
<tr>
<th>Operation</th>
<th>Big-O time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Insertion</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Removal</td>
<td>O(log n)</td>
</tr>
<tr>
<td>In-order iteration over all elements</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

Because these data structures are extremely popular in applications, it is important to investigate the ways of hastening basic functions. From this Point of view we are studying AVL and Red-black trees. Of course, details depend on a type of tree and differ, because they have different prototypes from operations research problems: how to choose the proper place for police, emergency hospital and school e.t.c.

“Center versus root” means that in some cases it is advantageous to change the return value of some tree functions from root to some dynamically and properly chosen node, called center. “Radius versus height” means that the maximal number of edges in paths from center to leaves can be made less than the height of tree, properly choosing center.

In our opinion, at least search time will be reduced in standard applications of BSTs such as OS kernels and other system software. Of course, complete conclusion can be made after sound tests. Note that empirical study of relationship between the algorithms used for managing BST-based data structures and performance characteristics in real systems is rather special and complex type of analysis (see [3]).

2 Visual interpretation.

Let us describe the need of the notion “center” on concrete samples of Red-black tree, appeared when calculating values of Ackerman function $A(3, 2)$, $A(3, 3)$.

It is seen from the following three pictures, where keys of red nodes are in brackets, that the existing explicit logarithmic dependence between the number of nodes and the height (see [1], taking into account that we do not consider NIL-nodes)

$$ (h + 1) \leq 2 \cdot \log_2 (n + 1), $$
is realistic, the height tends to vary in possible limits from $\lceil \log_2 n \rceil$ to $2 \cdot \log_2 (n+1) - 1$.

Pic. 1 shows, that root_key is 7, height is 6, number of nodes $n = 28$, so minimal possible value of the height is 4, and maximal possible value (by virtue of 1)) is 7. The real value of the height is 6, closely placed to maximal possible value in case $n = 28$.

Suppose we are able to make basic functions working with center and radius instead of root and height without less of efficiency. Then we can declare the node with key 15 to be center and its corresponding radius will be equal to 5 (really, in this case radius is 4, because only one path has 5 edges).

The above table clearly represents results of possible changes in handling trees – basic operations become faster.

### 3 Possible changes in codes.

We should estimate a price we will have to pay for extra efficiency – possible complications in program codes.

To this end, let us consider pseudo code of modified search function $C_{\text{Search}}(c, k)$, were $c$ is pointer to the center, and $k$ is the key of the node to be find. Usually, we assume that by $p(x)$, $l(x)$, $r(x)$ are denoted correspondingly parent, left-child, right-child of arbitrary node $x$.

Note that there are various modifications of BSTs depending on node representation. For node representation, parent pointers are generally fastest (see [3]). Therefore considering possible changes in program codes we assume BST representation with parent pointers.

```c
C_Search(c, k)
{
    if ( k < key(c) )
    {
        while (c==r(p(c)) && k<k(p(c)))
            c=p(c);
        return search(l(c), k);
    }
    if ( k >= key(c) )
    {
        while (c==l(p(c)) && k>=k(p(c)))
            c=p(c);
        return search(r(c), k);
    }
}
```

Here, we mean that $\text{root!} = r(p(\text{root}))$, $\text{root!} = l(p(\text{root}))$ and $\text{search}(x, k)$ is standard recursive search function, used in BSTs.
References:


