Splay Tree

Ravi Teja Cheruku
Indiana State University
Terre Haute,IN,USA

December 13, 2011

Abstract

Splay trees are self branching binary search tree which has the property of reaccessing the elements quickly that which are recently accessed. Splay trees have basic operations such as Insertion,Search,Deletion. The performance of the splay trees are much efficient than other search trees.

Contents

1 INTRODUCTION **2**

2 HISTORY **2**

3 ADVANTAGES **3**

4 DISADVANTAGES **3**

5 STATEMENT OF PROBLEM **3**

6 ALGORITHM **4**
6.1 Operations 4
6.1.1 When to Splay 4
6.1.2 Splaying 5
6.1.3 Zig Step 5
6.1.4 Zig-Zig step 6
6.1.5 Zig-Zag step 7
6.1.6 Results of Splaying 8
6.1.7 Insertion 8
6.1.8 Deletion 9

7 Splay Tree Application **9**

8 TIME COMPLEXITY **10**

9 Reference **11**

1 INTRODUCTION

Splaying is the basic operation for the splay trees which rearranges the tree so that element is placed at the root of the tree. The performance of the splay trees depends on the self balancing and self optimizing. The nodes of the tree are moved closer to the root so that they can be accessed quickly. In real time splay trees are used for implementing caches. The worst case with this splay tree algorithm is that this will sequentially access all the elements of the tree which makes the tree unbalanced. The reaccessing of the first element takes more time as it has to access all the remaning elements and then it starts accessing the first. if the access pattern is nonuniform, it needs an extra space for storage of balance information.

The real time applications of the splay trees are

- It is used to implement caches. Cache keeps track of the contents of memory locations that were recently requested by processor. It can be made to deliver requests much faster than main memory. Simiarly, splay trees uses this concept of accessing the elements quickly that which were recently accessed. Splay trees are used to implement Cache algorithms.

- it has the ability of not to store any data, which results in minimization of memory requirements.

- It can also be used for data compression, e.g. dynamic huffman coding.

The main idealogy behind selecting the splay tree is to reduce the consumption of time while reaccessing of the elements and to improve the performance and also to reduce the storage space.

2 HISTORY

Splay trees are invented by Daniel Dominic Sleator and Robert Endre Tarjan in 1985. Splay trees are self adjusting binary search tree , these trees perform better than other search trees. It performs basic operations such as insertion, search and deletion in amortized time.

3 ADVANTAGES

- It is easy to implement than other self balancing binary search trees,such as Red black trees or AVL trees. Much simpler to code than AVL, Red Black trees.

- Require less space as no balance information is required.

- Can be much more efficient if usage pattern is skewed. Good heuristics can have amortized running time similar to the running time of the best structure.

4 DISADVANTAGES

- More local adjustments during Search operations. Searching the key starts from the root node. If key is not found in the left node which has nodes less than the given key, then it searches in the right node which has nodes greater than the key. In order to splay the searched key to root of the tree more local adjustments has to be made.

- Individual operations can be expensive . drawback for real-time applications.

- The main disadvantage of splay trees is that the height of a splay tree can be linear. After accessing all n elements in non-decreasing order, Since the height of a tree corresponds to the worst-case access time, this means that the actual cost of an operation can be slow. However the amortized access cost of this worse case is logarithmic, O($log\ n$). Splay trees can change even when they are accessed in a $'$read-only$'$ manner. This complicates the use of splay trees in a multi-threaded environment. Specifically, extra management is needed if multiple threads are allowed to perform search operations.

5 STATEMENT OF PROBLEM

The disadvantage of the splay tree is the height. When we access all the elements of the tree in order which corresponds to the worst case access time, which menas the cost of each operation can be slow. I would like to solve this issue regarding the height of the tree which causes the tree to consume more time to access. Amortized cost for locating an item in the tree would be O($log\ n$),where

n is the number of items in the tree. The cost of an individual operation is O($log\ n$) on average, but an individual operation can be more expensive.

6 ALGORITHM

Splay trees are used to implement caches and also used for the minimization of memory requirements. Splay trees are self adjusting binary search trees which performs basic operations such as

- **Search**
- **Insertion**
- **Deletion**

Search,insert,delete operations are like in Binary Search trees, except at the end of each operation, a special step called Splaying is done. Splaying the tree rearranges the tree so that element is placed at the root of the tree. It uses tree rotations to bring the element to the top.

6.1 Operations

The main basic opeations of the Splay trees are

- **Search**
- **Insert**
- **Delete**
- **Splaying**

6.1.1 When to Splay

- **Search**: Splay node where key was found.
- **Insert**: When an item is inserted , a Splay is performed. As a result, the newly inserted node becomes the root of the tree.
- **Delete**: Splay parent of removed node which is either the node with the deleted key or its successor.

6.1.2 Splaying

To perform a splay operation , there is need to carry out sequence of Splay steps, which moves the node closer to the root. The recently accessed nodes are kept closer to the root so that tree remains balanced.

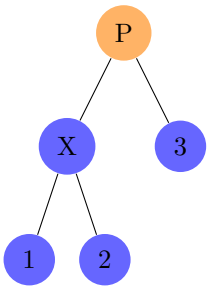
Each Splay step depends on three factors

- X is left or right child of its parent node,P.
- Check P is root node or not,if not
- P is left or right child of its parent G.

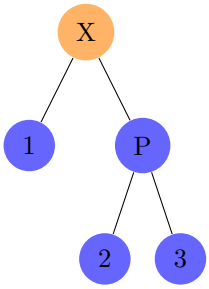
The three types of Splay steps are

6.1.3 Zig Step

- Let X be a non-root node on the access path on which we are rotating.
- If the parent of X is the root of the tree, we merely rotate X and the root.
- This is the same as a normal single rotation.

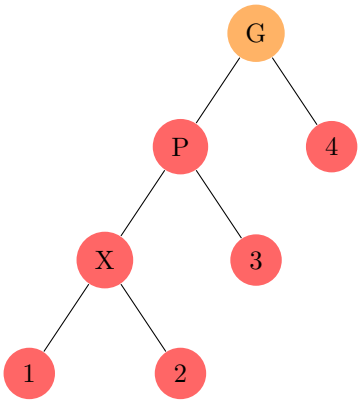


After performing the **Zig step** the tree is transformed with X as root and the tree is rotated as edge between X and P.

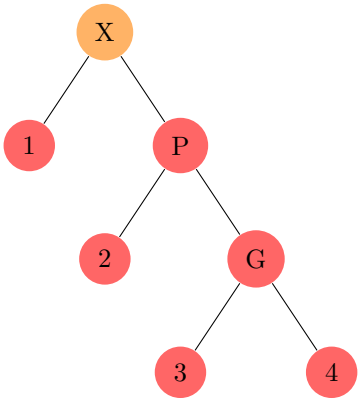


6.1.4 Zig-Zig step

- This is a different rotation from the previous one.
- Here X and P are either both left children or both right children.
- This is different from the rotate-to-root. Rotate-to-root rotates between X and P and then between X and G. The zig-zig splay rotates between P and G and X hand P.



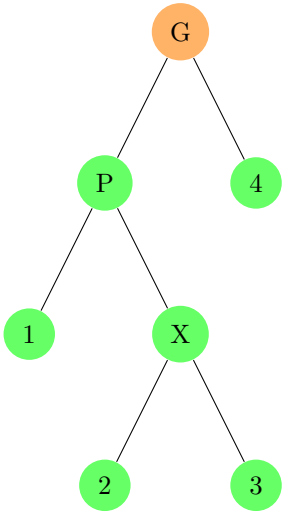
After **Zig-Zig step** is performed, X becomes the root for the transformed tree.Tree is rotated on the edge joining P and its parent G.



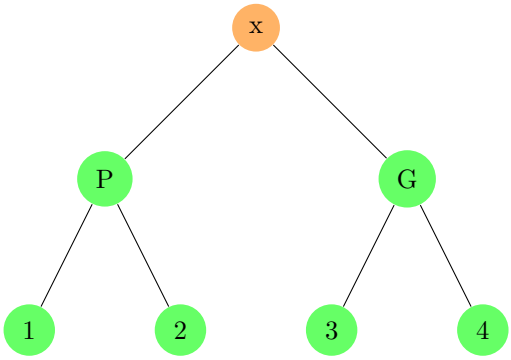
6.1.5 Zig-Zag step

- In this case, X and both a parent P and a grandparent G. X is a right child and P is a left child or vice versa.

- This is same as the double rotation.



After **zig-zag** step is performed, X becomes the root for the transformed tree. The tree is rotated on the edge between X and P , then rotated between X and G.



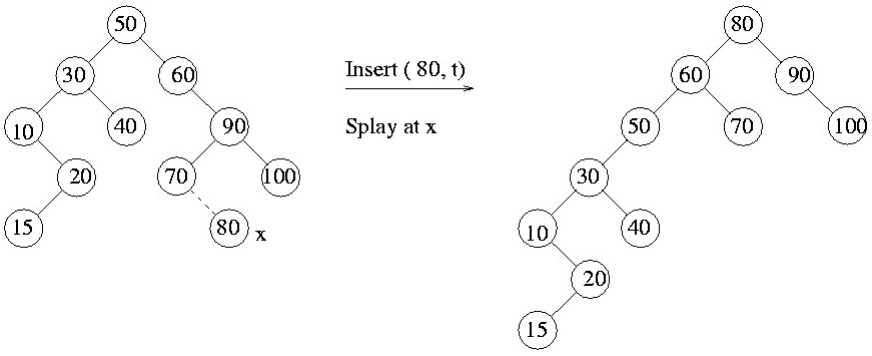
6.1.6 Results of Splaying

- The result is a binary tree, with the left subtree having all keys less than the root, and the right subtree having keys greater than the root.
- The resulted tree is more balanced than the original tree.
- If an operation near the root is done, the tree can become less balanced.

6.1.7 Insertion

To insert a node in to the tree

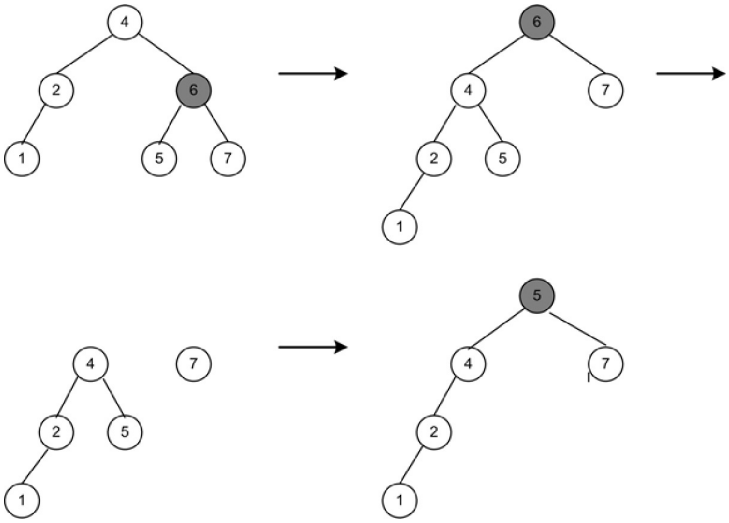
- Insert a node normally in to the tree.
- Splay the newly inserted node to the top of the tree.



6.1.8 Deletion

- Access the node to be deleted bringing it to the root.
- Delete the root leaving two subtrees L left and R right.
- Find the largest element in L, thus the root of L will have no right child.
- Make R the right child of L ′s root.

Delete element 6

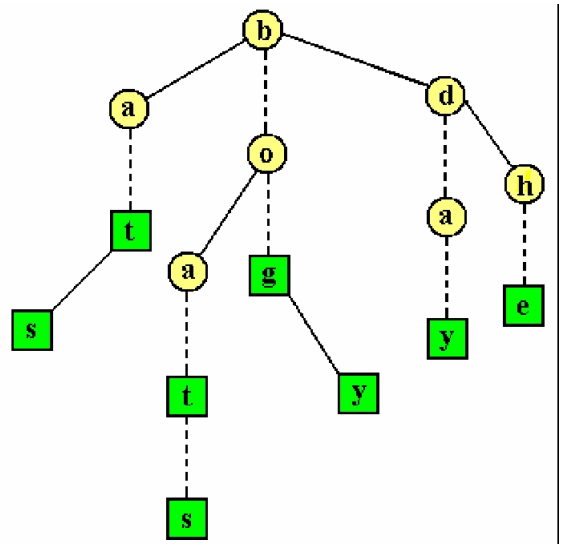


7 Splay Tree Application

In this section, an application for splay tree will be introduced. The application is **Lexicographic Search Tree**.

In a LST, all the circles and squares are the nodes of the tree. Each node will contain a character and the square nodes are the terminal nodes of the strings. Solid line edges form the splay tree, and the nodes connected by a solid line represents different strings. The dashed line edges used to represent a single string. From the figure, it stores different words "at", "as", "bat", "$bats$", "bog", "boy", "day" and "he". When a string is requested, we just splay the character in the string one by one. In LST, after the string is accessed, the

first character of the string will become the root, as a result the most frequent accessed string will be near the root.



8 TIME COMPLEXITY

For m operations to be performed on splay tree it requires O($mlog\ n$) steps where n is the size of the tree. Therefore the amortized complexity of the splay tree is O($log\ n$). The time complexity of maintaining a splay trees is analyzed using an Amortized Analysis. Each tree operation has actual costs proportional to its running time. Each splay operation has amortized cost O($log\ n$) thereby on an average the time complexity of each operation on the splay tree is

- **Search**: Searching for a node in the tree would take O($log\ n$).
- **Insert**: Inserting a node in to the tree takes O($log\ n$).
- **Delete**: Deleting a node from the tree takes O($log\ n$).

9 Reference

References

[1] Data Structures and Their Algorithms, Lewis and Denenberg, Harper Collins,

[2] Algorithms by S dasgupta U.V Vazirani

[3] wikipedia splaytree http://en.wikipedia.org/wiki/Splay_tree

[4] Self-Adjusting Binary Search Trees DANIEL DOMINIC SLEATOR AND ROBERT ENDRE TARJAN