1 Motivation

Binary search trees (BST) are meant to solve dynamic dictionary problem. n keys are stored in such a way that each access/insert/delete operation costs $O(\log n)$ time while still preserving the properties of BST. Balanced BSTs can achieve this goal but at a cost—they have to maintain extra "balance information". On the other hand, *splay trees* can adjust the structure of itself effectively so that all standard search tree operations still can have an *amortized time* bound of $O(\log n)$. By amortized time, we mean the time per operation averaged over a worst-case sequence of operations.

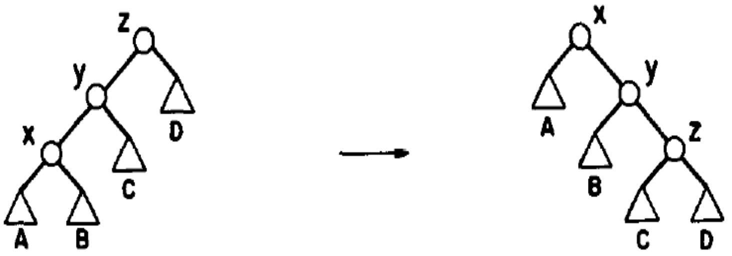
In all following discussion, we assume there are totally n distinct keys (hence n nodes in the BST) and m accesses. Here is the outline of this lecture note. In section 2, we define what is an *splay* operation and also introduces the concept of amortized time and potentail; in section 3, we discuss how the standard tree operations are to be implemented; in section 4, we prove some theorectical results concerning splay trees; finally, some possible variants of splay trees are discussed in section 5.

2 Splaying

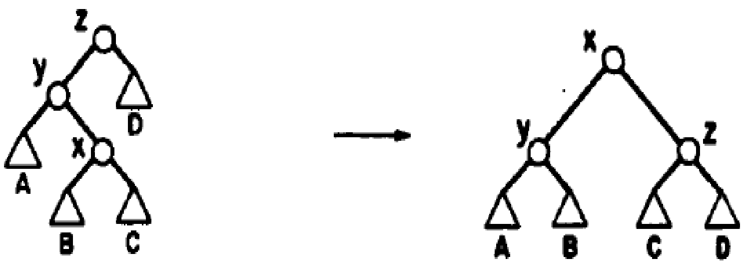
The splay operation, put in simplest form, not only accesses an specific key in the BST but also changes the tree structure dynamically. Specifically, a splay operation on node x is composed of the following three *splay steps*:

- Zig: If $p(x)$, the parent of x, is the root, rotate the edge joining x with $p(x)$.

- Zig-Zig: If $p(x)$ is not the root and x and $p(x)$ are both left or both right children, rotate the edge joining $p(x)$ with its grandparent $g(x)$ and then rotate the edge joining x with $p(x)$



- Zig-Zag: If $p(x)$ is not the root and x is a left child and $p(x)$ a right child, or vice-versa, rotate the edge joining x with $p(x)$ and then rotate the edge joining x with the new $p(x)$.



2.1 Amortized Time Complextiy & Potentail Function

To analyze the amortized complexity of splaying, we have to introduce a *potential function*. We assoicate a potential Φ with each possible configuration of the tree. Then, the *amortized time* a_i for i^{th} access is $a_i = t_i + \Phi_i - \Phi_{i-1}$, where t_i is the actual time. With this defintion, we can estimate the total of a sequence of m operations by

$$\sum_{i=1}^{m} t_i = \sum_{i=1}^{m} a_i + \Phi_0 - \Phi_m,$$

where Φ_0 is the initial potential and Φ_m is the final potential, and the difference between is called *potentail drop*. Note that the above equation can be directly interpreted as follows: *total access time = total amortized time + potential drop.*

Now we define how to decide the potential of a specific tree structure. Assuming that for each node x, we assign a weight $w(x) \in \mathcal{R}^+$. Let the *size* of a node x in the tree be $s(x) = \sum_{y \text{is decedant of } x} w(y)$ (note that x itself is included), and its corresponding *rank* be $r(x) = \log s(x)$, then the potential of the whole tree is $\Phi = \sum_{x \in BST} r(x)$.

2.2 The Amortized Time Complexity of Splaying

We introduce a very important Lemma.

Lemma 1 *(Access Lemma): the amortized time to splay a tree with root* t *at a node* x *is at most* $3(r(t) - r(x)) + 1 = O(\log(s(t)/s(x)))$.

We consider the three cases separately (please refer to figure in Section 1).

Zig: Amortized cost + potential drop =

$1 + (r'(x) + r'(y) - r(x) - r(y)$	note that only x, y change rank in the whole tree
$\leq 1 + r'(x) - r(x)$	since $r(y) \geq r'(y)$
$\leq 1 + 3(r'(x) - r(x))$	since $r'(x) \geq r(x)$

Zig-Zig: Amortized cost + potentail drop =

$2 + (r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$	note that only x, y, z change rank in the whole tree
$\leq 2 + r'(y) + r'(z) - r(x) - r(y)$	since $r(z) \geq r'(x)$
$\leq 2 + r'(x) + r(z) - 2r(x)$	since $r'(x) \geq r'(y)$ and $r(y) \geq r(x)$
$\leq 2(r'(x) - r(x)) + \log \frac{s(x)}{s'(x)} + \log \frac{s'(z)}{s'(x)} + 2$	
$\leq 2(r'(x) - r(x))$	since $\log \frac{s(x)}{s'(x)} + \log \frac{s'(z)}{s'(x)} \leq -2$
$\leq 3(r'(x) - r(x))$	

We utilize the concavity of log function to get the result that $\log \frac{s(x)}{s'(x)} + \log \frac{s'(z)}{s'(x)} \leq 2 * \log \frac{s(x)+s'(z)}{2*s'(x)} \leq 2 * \log(1/2) \leq -2$. (Note that $s(x) + s'(z) \leq s'(x)$)

Zig-Zag: Amortized cost + potentail drop =

$2 + (r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$	note that only x, y, z change rank in the whole tree
$\leq 2 + r'(x) + r'(z) - 2r(x)$	since $r'(x) \geq r'(z)$ and $r(x) \leq r(y)$
$\leq 3(r'(x) - r(x)) + r(x) + r'(z) - 2r'(x) + 2$	
$\leq 3(r'(x) - r(x)) + \log \frac{s(x)}{s'(x)} + \log \frac{s'(z)}{s'(x)} + 2$	
$\leq 3(r'(x) - r(x)) + (-2) + 2$	Again using the concavity of log function
$\leq 3(r'(x) - r(x))$	

Observe that only one zig step is necessary in the whole splay operation (this is where the 1 comes from), so the total amortized cost $\leq 3(r(t) - r(x)) + 1$. □

3 Implementing Standard Tree Operations on Splay Trees

We consider the following six operations.

1. access(i, t): if item i is in tree t, return a pointer to its location; otherwise, return a pointer to the null node.

2. insert(i, t): insert item i in tree t, assuming that it is not already there.

3. delete(i, t): delete item i from tree t, assuming that it is present.

4. join(t_1, t_2): combine tree t_1 and tree t_2 into a single tree containing all items from both trees and return the resulting tree. This operation assumes that all items in t_1 are less than all those in t_2 and destroyes both t_1 and t_2.

5. split(i, t): construct and return two tree t_1 and t_2, where t_1 contains all items in t less than or equal to i, and t_2 contains all items in t greater than i. This operation destroys t.

We know discuss how each operation is implemented.

- aceess: We search down the tree t, looking for i.
  - If node x containing i exists, splay on x.

  - If there is not any node containing i, splay on the last non-null node reached during the search.

- join: access the largest item i in tree t_1 and we will get another tree t_1' such that the root contains i and has a null right child. Attach t_2 to the root of t_1'.

- split: perform access(i, t) to get another new tree t'. And, depending on the value of its root,

  - If the root of t' is smaller than or equal to i, cut the right link of root of t'.
  - Otherwise, cut the left child of root of t'.

- insert: perform split(i, t) to get two trees t_1 and t_2 so that all items in t_1 is smaller than those in t_2. Create a new node x containing item i. Make a new tree t' whose root and left/right child are respectively x, t_1/t_2.

- delete: perform access(i, t) to get a new tree t', whose root is the node x containing i and whose left/right child are t_1/t_2. Delete node x. Perform the join(t_1, t_2) operation.

4 Complexity of Splay Trees

In this section, we present time complexity of splay tree under various access patterns. It is helpful to keep in mind that the total access time is composed of amortized time (which the Access Lemma in Section 2 can be utilized) and the potential drop.

The weight of the tree can be assigned in different ways. Access Lemma holds invariable irrespective of the weight assignment. But the upperbound of the potential drop over the sequence might be influenced: $\sum_{i=1}^{n} \log(W/w(i))$, where $W = \sum_{i=1}^{n} w(i)$.

Theorem 2 *(Balance Theorem). The total access time is $O((m+n)\log n + m)$.*

Proof: Set $w(i) = 1/n$. The amortized time of each access is $\leq 3\log 1/(1/n) + 1$. Potentail drop $\leq \sum_{i=1}^{n} \log(W/w(i)) = n(\log 1 - \log(1/n)) = n \log n$. Hence, the total time is $3m\log n + m + n\log n = O((m+n)\log n + m)$ □

Theorem 3 *(Static Optimality Theorem). If every item i is accessed at least $q(i)$ times, then the total accesss time is $O(m + \sum_{i=1}^{n} q(i)\log(m/q(i)))$.*

Proof: Set $w(i) = q(i)/m$. By access lemma, the amortized time of each access is $\leq 3 * \log 1/(q_i/m) + 1 = 3\log(m/q(i)) + 1$. As a result, the total amortized time is $\leq m + \sum_{i=1}^{n} q(i)(3 * \log(m/q(i)))$. The net potentail drop $\leq \sum_{i=1}^{n} \log(m/q(i))$. Hence the toal access time is $O(m + \sum_{i=1}^{n} q(i)\log(m/q(i)))$ □

Theorem 4 *(Static Finger Theorem). If f is any fixed item, the total acesss time is $O(n\log n + m + \sum_{i=1}^{m} \log(|i_j - f| + 1))$*

Proof: Set $w(i) = 1/(|i-f|+1)^2$. Then $W \leq \sum_{k=1}^{\infty} 1/k^2 = \pi^2/6 = O(1)$. Hence the amortized time of j^{th} access $\leq 2 * \log(|i_j - f| + 1) + 1$. The net potential drop $\leq \sum_{i=1}^{n} W/w(i) \leq n * \log O(1) - \sum_{i=1}^{n} \log 1/(|i-f|+1)^2 \leq n * \log O(1) - \sum_{i=1}^{n} \log 1/n^2 = O(n\log n)$. Therefore, the total access time is $O(n\log n + m + \sum_{i=1}^{m} \log(|i_j - f| + 1))$

5 Semi-adjusting Search Trees

In this section we discuss two possible variants of splay tree: semi-splaying trees. The first variant means that we only splay when the access path is "long" (will be defined later on); the second variant means that we stop splaying somethere in the m accesses.

Suppose every item i has a fixed weight $w(i)$, anad let c' be any constant greater than c. We call an access path for item i long if the depth d of the node containing i is at least $c' \log(W/w(i)) + c'/c$, and short otherwise.

Theorem 5 *(Long Splay Theorem). If we splay only when the access path is long and do not restructuring when the access path is short, then the total splaying time is $O(\sum_{i=1}^{n} \log(W/w(i)))$, that is, proportional to the amortized time to access each item once. The constant factor in this estimate is proportional to $c'/(c'-c)$. Thus the total restructuring time is independent of m, the number of accesses.*

Proof: Consider a node of depth $d \geq c' \log(W/w(i)) + c'/c$. Assume that the potential before and after are respectively Φ and Φ'. And we know that actual access time is d.

From Access Lemma, $d + \Phi' - \Phi \leq c \log(W/w(i)) + 1$. Thus, $d - c \log W/w(i) - 1 \leq \Phi - \Phi'$. If we devide both sides by d, we get

$$\frac{\Phi - \Phi'}{d} \geq 1 - \frac{c \log W/w(i) - 1}{d} \geq 1 - \frac{c \log W/w(i) + 1}{c' \log W/w(i) + c'/c} = 1 - c/c'.$$

We can get $\frac{d}{\Phi - \Phi'} \leq \frac{c'}{/} c' - c$, which means that the total splaying time over the sequence is proportional to at most $c'/(c'-c)$ times the overall potentail drop, which we know is $\sum_{i=1}^{n} \log(W/w(i))$. □

We now discuss another form semi-splay tree. The primary idea is that we may stop splaying after j^{th} access while still getting a well-balanced tree. Here we define the average access time of a tree is $\sum_{i=1}^{n} p_i d_i$, where d_i is the depth of the node containing item i and p_i is the probability that item i is accesses.

Theorem 6 *(Snapshot Theorem). Suppose we begin with any initial tree, carry out a sequence of m accesses, and randomly stop splaying just before access j, with probability $1/m$ for each possible value of j. Then the search tree resulting from the sequence of splying operations has an expected average access time of $O((n \log n)/m + \sum_{i=1}^{n} p_i \log(1/p_i))$.*

Proof: For any access sequence α, we define p_α as the probaiblity of α. Let $t_\alpha(\beta)$ be the total time required to carry out the sequence of accesses β using splaying, assuming that we have first performed the sequence α (also by splaying). Let T_α be the average access time of the tree formed by performing the sequence of accesses α using splaying.

We know $T_\alpha = \sum_{i=1}^{n} p_i t_\alpha(i)$, [1] which means that the quantity to be estimated is $\sum_{|\alpha i| \leq m} p_{\alpha i} t_\alpha(i)/m$. We have

$$\sum_{|\alpha|=m-1} p_\alpha t(\alpha) + \sum_{|\alpha i|=m} p_{\alpha i} t_\alpha(i) = \sum_{|\alpha i|=m} p_{\alpha i} t(\alpha i),$$

[1] This part I thoroughly followed the proof in the paper. However, the definition of *average access time* here is different from the one given previously $\sum_{i=1}^{n} p_i d_i$. About the discrepancy of the defitions, Professor Amit Chakrabarti explains that the authors here assume that they count the complexity of splaying but do not really *do* it

from which by induction on m, we get:

$$\sum_{|\alpha i| \leq m} \frac{p_{\alpha i} t_\alpha(i)}{m} = \sum_{|\alpha = m} \frac{p_\alpha t(\alpha)}{m}, \text{ for } m \geq 1.$$

Set $w(i) = p_i + 1/n$ for each item i, again utilitzing the Access Lemma, for any sequence α,

$$t(\alpha) = O(n \log n + m + \sum_{j=1}^{m} \log(\frac{1}{p_{\alpha(j)}})),$$

where $\alpha(j)$ is the itme accessed during access j in sequence α. Thus, the expected value of the average access time is

$$O(\frac{n \log n}{m} + \sum_{|\alpha|=m} p_\alpha (\sum_{j=1}^{m} \frac{\log(1/p_{\alpha(j)})}{m})).$$

Now consider all the terms in the double sum such that $\alpha(j) = i$. For any fixed access j, the sum of all terms is $p_i \log(1/p_i)/m$. As there are m possible values of j, the sum of all terms such that $\alpha(j) = i$ is $p_i \log(1/p_i)$. Thus, the expected average access time is $O((n \log n)/m + \sum_{j=1}^{n} p_i \log(1/p_i))$. $\square$