Splay Trees
Computer Science 3650

1. What is a splay tree?

A **splay tree** is a binary search tree that is kept balanced in an unusual way. It is based on the *move to front heuristic*: when you look something up, move it to a place where it will be found quickly the next time. In the case of a splay tree, each time a search ends at node u, node u is moved to the root of the tree by doing what is called a *splay*. There are no other rebalancing steps.

A splay is done on every operation, whether that operation is a lookup, an insertion or a deletion. As a consequence, the structure of the tree is constantly changing, even if you are only doing lookups.

Initially, that looks like it might be useful sometimes, but that it will not do well in the worst case. In fact, it achieves $O(\log n)$ *amortized* time per lookup, insertion or deletion *in the worst case*. Put another way, if you start with an empty tree and do n lookups, insertions and deletions, the total time for those n operations is $O(n \log n)$.

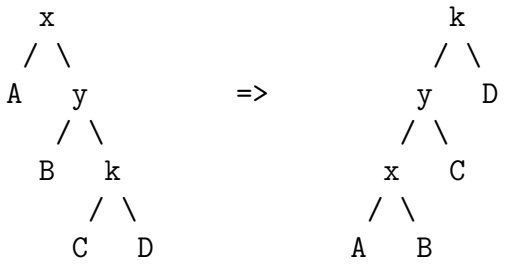
The reason that a splay tends to rebalance a tree is that it does more than move one value to the root. It also compresses the length of the path that it just traversed by roughly a factor of 2, which moves many nodes closer to the root than they were before.

2. Splaying a node to the root

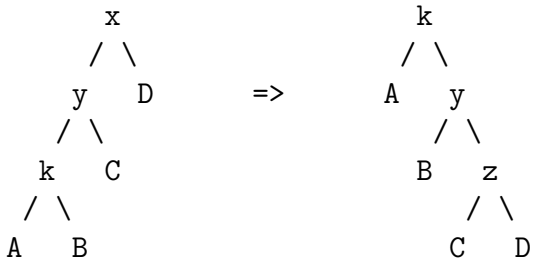
Suppose that a search ends at a node holding key k. To move k to the root, we use three operations.

2.1. Zig-Zig

A zig-zig operation moves a key k up two levels. It is done as follows, where x, y and k are keys and A, B, C and D are subtrees.



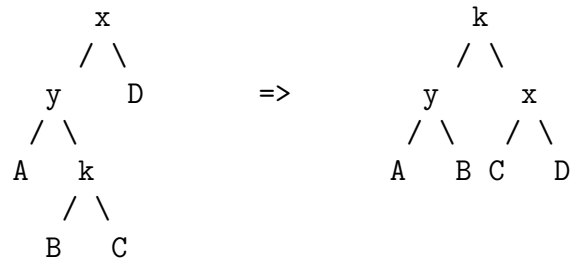
There is also a mirror image zig-zig, as follows.



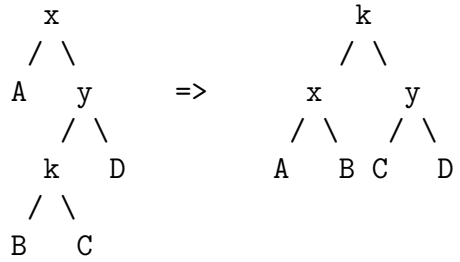
As you can see, a zig-zig is used when the node u that contains key k has a grandparent, and the path from u's grandparent to u takes two steps in the same direction, either both left or both right.

2.2. Zig-Zag

A zig-zag operation also moves a key k up two levels. It is done as follows, where x, y and k are keys and A, B, C and D are subtrees.



There is also a mirror image zig-zag, as follows.



As you can see, a zig-zag is used when the node u that contains key k has a grandparent, and the path from u's grandparent to u takes to steps in opposite directions, either both left-right or right-left.

2.3. Zig

A zig operation is only done at a node that does not have a grandparent. (That is, it is a child of the root.) A zig moves key k up one level, to the root, and is as follows, where x and k are keys and A, B and C are subtrees.

There is a mirror image operation that should be obvious.

3. Weight, rank and potential

We will do an analysis of splay tree operation using the physicists approach. That requires a few definitions.

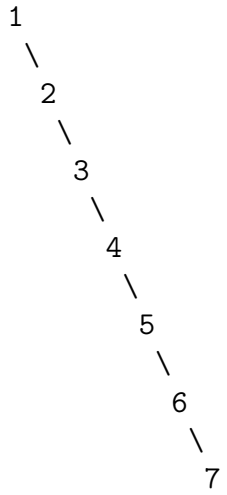
We will need to use logarithms to base 2. For brevity, we use $\log x$ to mean $\log_2 x$.

The **weight** $w(u)$ of a node u in a tree is the total number of nodes in the subtree rooted at u, counting u. Notice that a leaf has weight 1.

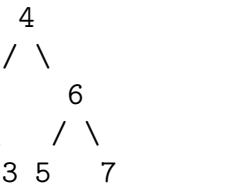
The **rank** of node u is $2\lfloor \log(w(u)) \rfloor$. We will write $r(u)$ for the rank of u. Since a leaf has weight 1, its rank is 0.

If t is a binary tree, the **potential** $\Phi(t)$ of t is the sum of the ranks of all of the nodes in t.

For example, suppose t is the following extremely unbalanced binary search tree.



Summing from the leaf up in t gives a potential of $2\lfloor\log(1)\rfloor+2\lfloor\log(2)\rfloor+2\lfloor\log(3)\rfloor+2\lfloor\log(4)\rfloor+2\lfloor\log(5)\rfloor+2\lfloor\log(6)\rfloor+2\lfloor\log(7)\rfloor = 0+2+2+4+4+4+4 = 20$. Now consider the following well balanced tree t'.



Its potential is $4(2\lfloor\log(1)\rfloor)+2(2\lfloor\log(3)\rfloor)+2\lfloor\log(7)\rfloor = 2+2+4 = 8$. Poorly balanced trees tend to have a high potential because of the deep nodes, which are part the subtrees of all of their ancestor nodes.

4. Some properties of ranks

Ancestor Lemma. Suppose that v is an ancestor of u. Then $r(v) \geq r(u)$.

Proof. This is clear since $w(v) > w(u)$. (Note that we cannot conclude that $r(v) > r(u)$ because u might be a leaf, and a leaf has rank 0.)

First Equal Ranks Lemma. Suppose that p is the parent of two nodes u and v, and that $r(u) = r(v)$. Then $r(p) > r(v)$.

Proof. Notice that u and v are nodes, not subtrees.

Case 1. Suppose $w(u) \geq w(v)$. Then

$$\begin{aligned} w(p) &= w(u) + w(v) + 1 \\ &\geq w(v) + w(v) + 1 \\ &> 2w(v). \end{aligned}$$

Since $\log(2w(v)) = \log(w(v))+1$, it must be the case that $\lfloor \log(2w(v)) \rfloor > \lfloor \log(w(v)) \rfloor$. So

$$\begin{aligned} r(p) &= 2\lfloor \log(w(p)) \rfloor \\ &\geq 2\lfloor \log(2w(v)) \rfloor \\ &> 2\lfloor \log(w(v)) \rfloor \\ &= r(v) \end{aligned}$$

and we have proved the lemma for this case.

Case 2. Suppose $w(v) \geq w(u)$. That is not a problem. We have just shown that $r(p) > r(v)$. But the lemma supposes that $r(u) = r(v)$, so surely $r(p) > r(u)$.

Second Equal Ranks Lemma. Suppose that p is the parent of two nodes u and v, and that $r(u) = r(v)$. Then $r(p) \geq r(v) + 2$.

Proof. We have shown that $r(p) > r(v)$. But the rank of a node x is defined to be $2\lfloor \log(w(x)) \rfloor$. Clearly, ranks are always even integers. For even integers i and j, if $i > j$, then $i \geq j + 2$.

5. Analysis of zig, zig-zig and zig-zag operations

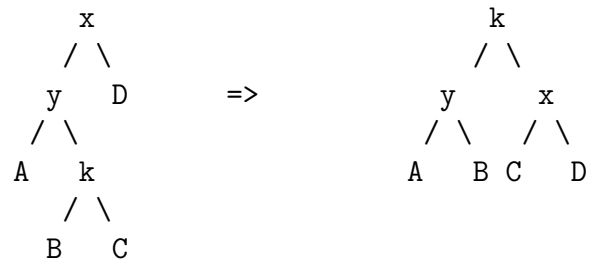
For simplicity, we only analyze a sequence of lookups. Analysis for inserts and deletes follow similar lines.

We will say that a zig has time-cost 1 because, excluding the key that it moves to the root, it looks at 1 node. Zig-zig and zig-zag operations cost 2 each.

It is convenient to refer to a node by the key that it contains. That is, a node that contains key x is simply called node x. Since all keys in a binary search tree are different, there is no ambiguity in that.

The **charge** for an operation is its cost plus the amount that it increases the potential.

Consider a zig-zag operation



Say that $r(x)$ is the rank of x in the left-hand tree above and $r'(x)$ is the rank of x in the right-hand tree, and do similarly for y and k.

Zig-Zag Lemma. The charge for the above zig-zag operation is no more than $2(r'(k) - r(k))$.

Proof. Trees A, B, C and D are unchanged, and their contribution to the potential is the same in both trees. So the change in potential is only influenced by nodes x, y and k. The charge for this zig-zag operation is $\mathcal{C}$ where

$$\mathcal{C} = 2 + r'(x) + r'(y) + r'(k) - r(x) - r(y) - r(k).$$

But node x in the left-hand subtree has the same weight as node k in the right-hand subtree. So $r(x) = r'(k)$, and those terms cancel, giving

$$\mathcal{C} = 2 + r'(x) + r'(y) - r(y) - r(k).$$

By the ancestor lemma, $r(y) \geq r(k)$. Replacing $r(y)$ by $r(k)$ gives

$$\mathcal{C} \leq 2 + r'(x) + r'(y) - 2r(k). \tag{1}$$

From this point, there are two cases.

Case 1. Suppose $r'(x) = r'(y)$. By the second equal rank lemma, $r'(k) \geq r'(x) + 2$. Replacing $r'(x)$ by $r'(k) - 2$ in (1) gives

$$\mathcal{C} \leq 2 + r'(k) - 2 + r'(y) - 2r(k).$$

By the ancestor lemma, $r'(k) \geq r'(y)$. Replacing $r'(y)$ by $r'(k)$ gives

$$\mathcal{C} \leq 2r'(k) - 2r(k).$$

We have proved this lemma for the case where $r'(x) = r'(y)$.

Case 2. Suppose $r'(x) > r'(y)$. Because ranks are even integers, it must be the case that $r'(x) \geq r'(y) + 2$. Replacing $r'(y)$ by $r'(x) - 2$ in (1) gives

$$\mathcal{C} \leq 2r'(x) - 2r(k).$$

By the ancestor lemma, $r'(k) \geq r'(x)$. So

$$\mathcal{C} \leq 2r'(k) - 2r(k).$$

We have proved this lemma for the case where $r'(x) > r'(y)$.

Case 3. Suppose $r'(y) > r'(x)$. This case holds by symmetry with case 2.

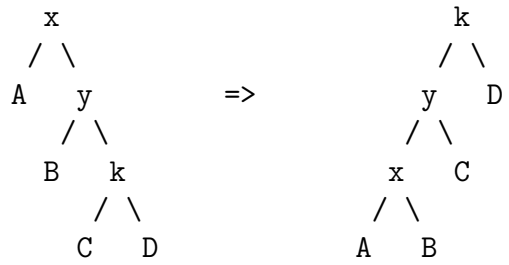
That completes the proof of the zig-zag lemma.

Weakened zig-zag lemma The charge for the zig-zag operation in the zig-zag lemma $3r'(k) - 3r(k)$.

Proof. It should be clear that $r'(k) \geq r(k)$, so $r'(k) - r(k) \geq 0$. So $2(r'(k) - r(k)) \leq 3(r'(k) - r(k))$.

The following two lemmas are proved using similar ideas.

Zig-Zig Lemma. Consider the following zig-zig operation.



The charge for this operation is no more than $3r'(k) - 3r(k)$.

Zig Lemma. Consider the following zig operation.

The charge for this operation is no more than $1 + r'(k) - r(k)$, which is less than or equal to $1 + 3(r'(k) - r(k))$.

6. Analysis of the charge for a splay

Theorem. If a splay is done on a tree that has n nodes, then the charge of that splay is at most $1 + 6\log(n)$.

Proof. To splay k to the root, we do a sequence of m zig-zigs and zig-zags, plus possibly one zig at the end, for some m. Let $r_0(k)$ be the rank of k before the splay starts, and let $r_i(k)$ be the rank of k just after the i-th step, for $i = 1, \ldots, m$. Then the total charge $\mathcal{C}$ excluding the final zig is at most

$$\mathcal{C} \leq 3(r_1(k) - r_0(k) + r_2(k) - r_1(k) + ... + r_m(k) - r_{m-1}(k)).$$

Almost all of the terms cancel, and we are left with a charge of

$$\begin{aligned} \mathcal{C} &\leq 3(r_m(k) - r_0(k)) \\ &\leq 3r_m(k) \end{aligned}$$

Now add the charge for one zig.

$$\begin{aligned} \mathcal{C} &\leq 3r_m(k) + 1 + 3r_{m+1}(k) - 3r_m(k) \\ &\leq 1 + 3r_{m+1}(k) \end{aligned}$$

At the end of the splay, k is at the root, so $w(k) = n$ and $r_{m+1}(k) \leq 2\log(n)$, and

$$\mathcal{C} \leq 1 + 6\log(n).$$

Theorem. A sequence of n lookups in a tree with n nodes has total time-cost $O(n \log n)$.

Proof. The time-cost is the total charge minus the total potential increase.

The sum of the charges is no more than $n(1 + 6\log n) = O(n \log n)$.

Keep in mind that the potential increase might be negative; the potential might have decreased.

We can get an upper bound on the potential of a tree that has n nodes. Every node must have weight at most n, since a subtree cannot have more nodes than the whole tree. That means every node has a rank no more than $2\log n$, and the potential of the tree is at most $2n \log n$.

If the tree starts with potential $2n \log n$ and ends with potential 0, then the total potential decrease is $2n \log n = O(n \log n)$. The potential cannot have decreased more than that.