10 Splay trees

Another approach to efficient search trees is to use a heuristic which doesn't make every operation efficient, but is guaranteed to keep the total cost, of a series of operations, low.

The trees in this section are called *splay trees* that it is easier to write code for splay trees than for red-black trees — and therefore it is easier to write correct code for splay trees. This is not much of a recommendation, and I have heard that splay trees operations are slow in practice. But the analysis of splay trees is (in this module) new and interesting.

This is an *amortised analysis.* There are two styles of amortised analysis. In this section it will use a *potential function* Φ.

This is a function of the totality of nodes (created during a series of operations). Our analysis expects the total number of nodes, n, is given in advance, that initially the system is a forest with no links, and at any time it is a forest composed of these n nodes. The initial potential is Φ_0 (actually, it is zero).

Suppose that the $c_1, \ldots$ are the actual costs of a series of operations. Ways to measure these costs will be considered below. The *amortised cost* of the i-th operation is *defined as*

$$c_i + \Phi_i - \Phi_{i-1}.$$

The trees will be modified using three kinds of operation, called *splay operations.* They preserve inorder, of course. They are illustrated in Figure 1.

The operations are to 'splay from p,' in three cases. They move p closer to the root. There is nothing to do if p is at the root. In any operation, we estimate the actual cost by the number of rotations.

- **Zig.** p has a parent q, and q is the root. Rotate p up, so p becomes the root.
- **ZigZig.** p has a parent q and a grandparent r, and p and q are both left children, or they are both right children. Rotate q up, then rotate p up.
- **ZigZag.** As for ZigZig, except that p is a right child and q a left child (as illustrated), or vice-versa. Rotate from p, and rotate from p again.

Now to define Φ.

- First, for each node x, we assume there is a 'weight' $w(x)$ assigned at the beginning. The only weighting we will consider is uniform: every node has weight $1/n$.

  The weight of nodes never changes.
- The rank rank(x) of a node x, which does change, is $\log_2 X$, where X is the total weight of all descendants of x, including x itself. Here the exact logarithm is used, not a rounded version.
- The potential Φ (at any point in the operations) is the sum of the node ranks at that time.

Recall that the amortised cost of the i-th operation is

$$c_i + \Phi_i - \Phi_{i-1}.$$

We estimate the amortised cost of a single splay operation where a node p is moved closer to the root. Let $\text{rank}(x)$ and $\text{rank}'(x)$ be the rankof nodes x before and after the splay, respectively.

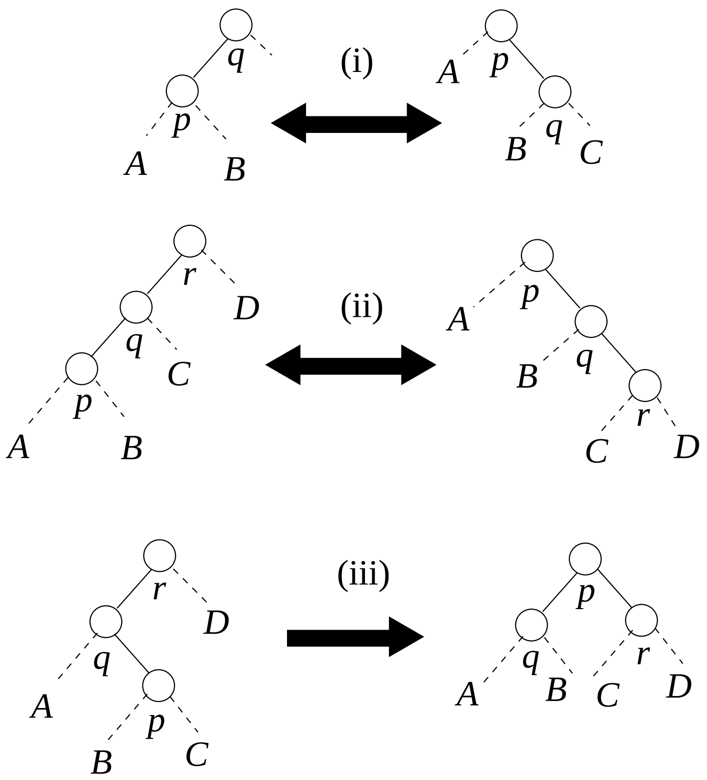

Figure 1: Splay operations: (i) zig, (ii) zigzig, (iii) zigzag. The first two work in both directions. In (iii) the case where p is the left child of a right child, is omitted.

(10.1) Lemma *A Zig operation has amortised cost* $\leq 1 + \text{rank}'(p) - \text{rank}(p)$.

Proof. This is case (i) in Figure 1. The ranks of nodes within the subtrees A, B, C do not change, the cost is 1 (rotation), and the amortised cost is

$$1 + \text{rank}'(p) + \text{rank}'(q) - \text{rank}(p) - \text{rank}(q)$$

Now, $\text{rank}'(q) < \text{rank}(q)$, so this is at most

$$1 + \text{rank}'(p) - \text{rank}(p)$$

and $\text{rank}'(p) > \text{rank}(p)$, so this is at most

$$1 + 3(\text{rank}'(p) - \text{rank}(p)). \quad \blacksquare$$

(10.2) Lemma *A ZigZag operation has amortised cost at most*

$$3(\text{rank}'(p) - \text{rank}(p)).$$

Proof. This is case (iii) of Figure 1. Let X be the weight of the new subtree at q, i.e., the total weight of q, A, and B. Let Y be the total weight of r, C, and D.

Ranks are unchanged within A, B, C, D, and at all nodes outside the illustrated subtrees (r is not necessarily the root). They change only at p, q, and r.

The actual cost is 2 rotations.

The amortised cost is

$$2 + \text{rank}'(p) + \text{rank}'(q) + \text{rank}'(r) - \text{rank}(p) - \text{rank}(q) - \text{rank}(r).$$

The claim is that this is at most $3\text{rank}'(p) - 3\text{rank}(p)$, or, equivalently

$$2\text{rank}'(p) - \text{rank}'(q) - \text{rank}'(r) - 2\text{rank}(p) + \text{rank}(q) + \text{rank}(r) \geq 2$$
$$2\text{rank}'(p) - \log_2 X - \log_2 Y + [\text{rank}(q) + \text{rank}(r) - 2\text{rank}(p)] \geq 2.$$

Now q and r are both ancestors of p before the splay, so the term $[\ldots]$ in square brackets is nonnegative (actually, positive). Also, $\text{rank}'(p) \geq \log_2(X+Y)$. So it is enough to prove

$$2\log_2(X+Y) \geq 2 + \log_2 X + \log_2 Y$$
$$(X+Y)^2 \geq 4XY$$

which is true, since $(X-Y)^2 \geq 0$. ∎

(10.3) Lemma *A Zig operation has amortised cost at most* $3(r'(p) - r(p))$.

Proof. As for the case (iii), we need to prove the following.

$$2\text{rank}'(p) - \text{rank}'(q) - \text{rank}'(r) - 2\text{rank}(p) + \text{rank}(q) + \text{rank}(r) \geq 2$$

This time X is the descendants of p before the splay, and Y is the descendants of r after the splay: $\log_2 X = \text{rank}(p)$ and $\log_2 Y = \text{rank}'(r)$. We aim to prove

$$2\text{rank}'(p) - \text{rank}'(q) - \text{rank}'(r) - 2\text{rank}(p) + \text{rank}(q) + \text{rank}(r) \geq 2$$

and it is enough to show

$$2\log_2(X+Y) - \text{rank}'(q) - \log_2(Y) - (\log_2 X + \text{rank}(p)) + \text{rank}(q) + \text{rank}(r) \geq 2$$

One occurrence of $\text{rank}(p)$ has been replaced by $\log_2(X)$.

$$2\log_2(X+Y) - \log_2 X - \log_2 Y - \text{rank}'(q) - \text{rank}(p) + \text{rank}(q) + \text{rank}(r) \geq 2$$

As previously $2\log_2(X+Y) - \log_2 X - \log_2 Y \geq 2$. So it is enough to show

$$-\text{rank}'(q) - \text{rank}(p) + \text{rank}(q) + \text{rank}(r) \geq 0$$

But $\text{rank}(p) < \text{rank}(q)$ and $\text{rank}'(r) - \text{rank}'(q) = \text{rank}'(p) - \text{rank}'(q) > 0$. ∎

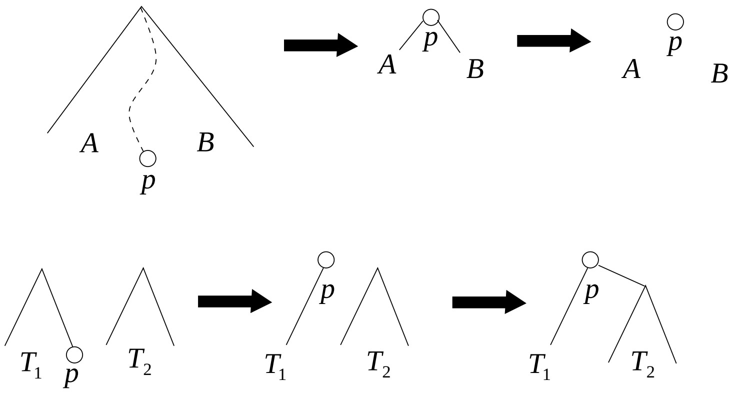

Figure 2: Split and join.

(10.4) Lemma *The amortised cost of bringing a node p to the root of a tree by repeated splaying is at most*

$$1 + 3(\text{rank}'(p) - \text{rank}(p))$$

where rank *is the rank function before the operations and* rank$'$ *is the rank afterwards.* ▮

Uniform weighting. We assume that the node weights are all $1/n$.

The rank of p at any time is $\log_2(d/n)$ where d is the number of descendants. Therefore the amortised cost of bringing p to the root by splaying is $\leq 1 + 3\log_2 n$.

10.1 Split, join, insert, and delete

To split a tree from a node p,

- bring p to the root by repeated splaying, so the tree has root p with left and right subtrees A and B.
- Then cut the links joining p to these subtrees. The actual cost of cutting the links is negligible.

To join two trees T_1 and T_2,

- if either is empty there is nothing to do. Otherwise,
- find the rightmost node p in T_1, and bring it to the root by splaying.
- Make T_2 the right subtree at p, at neglible actual cost.

To insert a new key k in a tree, search for the key in the usual way; if absent create and add a new node p in the usual way; and bring p to the root by splaying.

To delete a key k, search for it. If found at a node p, split the tree at T and join the two halves, consisting of the keys $< k$ and the keys $> k$.