6.854 Problem Set 2

Nada Amin
namin@mit.edu

September 19, 2007

Problem 1

The SUCCESSOR takes as input a pointer to the root of a tree and restructures the tree so that the successor of the root becomes the root. The operation does not change the tree if the root is already the rightmost element of the tree. The new root is returned.

Using the SPLAY operation as a subroutine, I implement the SUCCESSOR operation by traversing the tree, going right once from the root, and then left until a node with no left child is reached. This node is the successor, so I SPLAY it to the root, returning the new root.

Part (a)

The access lemma states that the amortized time to splay a tree with root t at a node x is $O(\log(s(t)/s(x)))$, where $s(x)$ is the sum of the weights of the descendants of node x. If I choose the weight $\frac{1}{n}$ for each node, it follows that the amortized time to splay a tree is $O(\log(n))$. The initial traversing of the tree doesn't cost more than the splaying, so SUCCESSOR has an amortized cost of $O(\log(n))$.

Part (b)

The scanning theorem states that given an aribitrary n-node splay tree, the total time to splay once at each of the nodes, in symmetric order, is $O(n)$. When the SUCCESSOR function is applied repeatedly until it finds the rightmost node, it is as if we're splaying a tree of m nodes, the nodes on the right side of the root, in symmetric order. So, by the scanning theorem, the repeated application of SUCCESSOR is $O(m)$ where m is the number of nodes initially on the right side of the root. Since we repeatedly apply SUCCESSOR m times, the amortized running time of one SUCCESSOR operation is $O(1)$.

Problem 2

I show that for $n \geq 4$, it is possible to restructure any binary tree on n nodes into any other binary tree on n nodes by a sequence of SPLAY operations.

The proof is by induction.

For the base case, $n = 4$, I turn the tree into a left path by splaying the nodes in order. From this left path, I can get to any other 4-nodes binary tree configuration. There are 14 such configurations. I leave the details of the enumeration out for brevity.

For the inductive case, it suffices to show that I can make a leaf out of any node of a tree with $n \geq 4$ nodes. If this node is a leaf in the final configuration, then I can inductively restructure the other $n - 1$ nodes by splaying without affecting this leaf and without the leaf affecting the rest of the splaying (this is obvious from the way splaying operates). This inductive process will result in the desired tree because no tree differs just in the position of one leaf node. Indeed, if this were the case, it would mean that the path to this leaf was different in the two trees: so, in one tree, I would turn right at a node, when in the other, I would turn left. But that would mean, that the leaf were simultaneously smaller and greater than another node, which would be a contradiction.

It remains to show that I can make a leaf out of any node of a tree with $n \geq 4$ nodes. If the node we want to turn into a leaf is the smallest, we can make it a leaf by splaying on it first and then its successors in order. If it is the largest, we do the opposite: splay on the predecessors, from smallest to largest, and finally on the node to be a leaf. If the node is the second smallest, we splay the successor of its successor, its successor, the node itself, its predecessor, and the successor of its successor again. Otherwise, we splay the nodes in order (from smallest to largest), and then on the predecessor of its predecessor once more.

Problem 3

Part (a)

I argue that along a given search path, there can be at most $O(\log n)$ balanced triples. Indeed, if a triple is balanced, that means that any node below x must have $< \frac{9}{10}d$ nodes below it, where d is the number of descendants of z (the grand-father of x). At the start, $d = n - 1$, so there must be $< \frac{9}{10}(n-1)$ nodes below the first balanced triple in the search path. Similarly, after the k^{th} balanced triple, there must be $< (\frac{9}{10})^k n$ nodes:

$$\begin{aligned}&\frac{9}{10}(\cdots\frac{9}{10}(\frac{9}{10}(n-1)-1)\cdots-1)\\&=(\frac{9}{10})^k n-\sum_{i=1}^{k}(\frac{9}{10})^i\leq(\frac{9}{10})^k n\end{aligned}$$

We're looking for the largest possible number of balanced triples in a given search path, that is, the largest possible k. So we solve:

$$\begin{aligned}(\frac{9}{10})^k n &\leq 1\\ k\log(\frac{9}{10})+\log(n)&\leq 0\\ k\leq\frac{\log(n)}{\log(\frac{10}{9})}&=O(\log n)\end{aligned}$$

Hence, along a given search path, there can be at most $O(\log n)$ balanced triples.

Part (b)

I argue that when a biased triple is rotated, the potential decreases by a constant paying for the rotation.

ZIG-ZIG

$|A|, |B|, |C|, |D|$ refer to the subtrees shown in Figure 1.

For a biased triple, we know that

$$\begin{aligned}\frac{9}{10}(|A|+|B|+|C|+|D|+2)&\leq|A|+|B|\\ \frac{1}{10}(|A|+|B|+|C|+|D|+2)&>|C|+|D|\end{aligned}$$

Figure 1: ZIG-ZIG

$s(x)$ is the size of the subtree rooted at node x before the ZIG-ZIG, and $s'(x)$ after the ZIG-ZIG. $r(x) = \log(s(x))$ and $r'(x) = \log(s'(x))$.

$$\begin{aligned} s(x) &= |A| + |B| + 1 \\ s'(x) &= |A| + |B| + |C| + |D| + 3 \\ s'(x) &= s(x) + 2 + |C| + |D| \\ s'(x) &< s(x) + 2 + \frac{1}{10}(|A| + |B| + |C| + |D| + 2) \\ s'(x) &< s(x) + 2 + \frac{1}{10}s'(x) \\ \frac{9}{10}s'(x) &< s(x) + 2 \\ r'(x) - r(x) &< \log 10 - \log 9 \end{aligned}$$

Thus, the rank of x increases by at most $\log 10 - \log 9$.

$s(y)$ is the size of the subtree rooted at node y before the ZIG-ZIG, and $s'(y)$ after the ZIG-ZIG. $r(y) = \log(s(y))$ and $r'(y) = \log(s'(y))$.

$$\begin{aligned} s(y) &= 2 + |A| + |B| + |C| \\ s'(y) &= 2 + |B| + |C| + |D| \\ s'(y) - s(y) &= |D| - |A| \end{aligned}$$

Thus, the worst-case for y happens when $|D|$ is maximum (that is, $|C| = 0$) and $|A| = 0$. But then,

$$\begin{aligned} s'(y) - s(y) &< \frac{1}{10}(|A| + |B| + |C| + |D| + 2) \\ s'(y) - s(y) &< \frac{1}{10}(|B| + |C| + |D| + 2) \\ s'(y) - s(y) &< \frac{1}{10}s'(y) \\ \frac{9}{10}s'(y) &< s(y) \\ r'(y) - r(y) &< \log 10 - \log 9 \end{aligned}$$

Figure 2: ZIG-ZAG

Thus, the rank of y increases by at most $\log 10 - \log 9$.

$s(z)$ is the size of the subtree rooted at node z before the ZIG-ZIG, and $s'(z)$ after the ZIG-ZIG. $r(z) = \log(s(z))$ and $r'(z) = \log(s'(z))$.

$$\begin{aligned} s(z) &= |A| + |B| + |C| + |D| + 3 \\ s'(z) &= |C| + |D| + 1 \\ s'(z) - 1 &< \frac{1}{10}(s(z) - 1) \\ r'(z) - r(z) &< \log 1 - \log 10 \end{aligned}$$

Thus, the rank of z decreases by at least $\log 1 - \log 10$.

Putting all this togeter, I conclude that the potential decreases when rotating a biased triple using a ZIG-ZIG:

$$\begin{aligned} & r'(x) - r(x) + r'(y) - r(y) + r'(z) - r(z) \\ = & \log 10 - \log 9 + \log 10 - \log 9 + \log 1 - \log 10 \\ = & \log 10 - 2\log 9 = \log 10 - \log 18 < 0 \end{aligned}$$

ZIG-ZAG

$|A|, |B|, |C|, |D|$ refer to the subtrees shown in Figure 2.

For a biased triple, we know that

$$\begin{aligned} \frac{9}{10}(|A| + |B| + |C| + |D| + 2) &\leq |B| + |C| \\ \frac{1}{10}(|A| + |B| + |C| + |D| + 2) &> |A| + |D| \end{aligned}$$

I use the same reasoning as for the ZIG-ZIG to conclude that the rank of x increases by at most $\log 10 - \log 9$.

For y,

$$\begin{aligned} s(y) &= |A| + |B| + |C| + 2 \\ s'(y) &= |A| + |B| + 1 \\ s(y) - s'(y) &= |C| + 2 \end{aligned}$$

The rank of y decreases. The smallest decrease happens when $|C| = 0$, in which case the rank stays about the same.

For z,

$$\begin{aligned} s(z) &= |A| + |B| + |C| + |D| + 3 \\ s'(z) &= |C| + |D| + 1 \\ s(z) - s'(z) &= |A| + |B| + 2 \end{aligned}$$

The rank of z decreases. The smallest decrease happens when $|A| = |B| = 0$, in which case the rank stays about the same.

However, because of the constraints on a biased triple, we cannot have $|B| = 0$ and $|C| = 0$ simultaneously. So the rank of either y or z will have to decrease significantly as a result.

If I let $|B| = 0$, then the rank of y will decrease by at least $-\log 10$:

$$\begin{aligned} s(y) - s'(y) &\geq \frac{9}{10}(|A| + |B| + |C| + |D| + 2) \\ s(y) - s'(y) &\geq \frac{9}{10}(|A| + |B| + |C| + 2) \\ s(y) - s'(y) &\geq \frac{9}{10}s(y) \\ \frac{1}{10}s(y) &\geq s'(y) \\ r(y) - r'(y) &\geq \log 10 - \log 1 \end{aligned}$$

Similarly, if I let $|C| = 0$ and $|A| = 0$, then the rank of z will decrease by at least $-\log 10$.

Thefore, in either case, I can conclude that the potential decreases by rotating a biased triple using a ZIG-ZAG, since

$$\log 10 - \log 9 - \log 10 = \log 9 < 0$$

I have thus shown that both ZIG-ZIG and ZIG-ZAG rotations on biased triples decrease the potential.

Part (c)

I argue that when a balanced triple is rotated, the potential increases by at most $2(r(z) - r(x))$.

ZIG-ZIG

$$\begin{aligned} r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) & \quad \text{since } r'(x) = r(z) \\ = r'(y) + r'(z) - r(x) - r(y) & \quad \text{since } r'(x) \geq r'(y) \text{ and } r(y) \geq r(x) \\ \leq r'(x) + r'(z) - 2r(x) & \quad \text{since } r'(x) = r(z) \text{ and } r'(z) < r(z) \\ \leq 2(r(z) - r(x)) & \end{aligned}$$

ZIG-ZAG

$$\begin{aligned} r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) & \quad \text{since } r'(x) = r(z) \text{ and } r(x) \leq r(y) \\ \leq r'(y) + r'(z) - 2r(x) & \quad \text{since } r'(y) < r(z) \text{ and } r'(z) < r(z) \\ \leq 2(r(z) - r(x)) & \end{aligned}$$

Part (d)

From part (b), I conclude that enough potential falls out of the system to pay for all the biased rotations.

In part (c), I showed that a balanced rotation costs $\leq 2(r(z) - r(x)) \leq 2\log\frac{s(z)}{s(x)}$. From part (a), I know that $s(x) \leq \frac{9}{10}s(z)$ for a balanced triple. Let's consider the two extreme cases:

if $s(x) = \frac{9}{10}s(z)$, then the potential increases by a constant $2\log\frac{10}{9}$, but there can be $O(\log n)$ such increases.

if $s(x) = 1$ **and** $s(z) = n$, then the potential increases by $O(\log n)$ but there can only be 1 such increase.

So the size of $s(x)$ limits either the number of possible balanced triples if small or the increase in potential if large. Therefore, rotating the balanced triples costs at most $O(\log n)$.

Therefore, the real work and amount of potential introduced by the balanced rotation is $O(\log n)$, which thus bounds the amortized cost.

Part (e)

I prefer the analysis presented in class, because it is more elegant and more general, since the access lemma allowed us to make a series of interesting conclusions about splay trees.

Problem 4

Part (a)

Supposing a random function is used to map each item to buckets, I give a good upper bound on the expected number of collisions.

Before the k^{th} item is inserted, at most $\frac{k-1}{n^{1.5}}$ buckets are used. Hence, the probability that the bucket where we will insert the k^{th} item is used in each array is $\leq (\frac{k-1}{n^{1.5}})^2$.

By linearity of expectations, the expected number of collisions is bounded by

$$\sum_{k=1}^{n}(\frac{k-1}{n^{1.5}})^2 = \sum_{k=0}^{n-1}(\frac{k}{n^{1.5}})^2 = \frac{(n-1)(n)(2n-1)}{6n^3} \leq \frac{1}{3}$$

Part (b)

My reasoning of part (a) requires only that the hash function be pairwise-independent, and that the hash functions be independent (in the case where each array uses a different function). So I can use two independent 2-universal hashing functions, one for each random function mapping each item to buckets.

Part (c)

The algorithm is simple: randomly generate a pair of 2-universal hashing functions until finding one which produces no collisions on the set of n items.

Since I determined in part (a) that the expected number of collisions is $\leq \frac{1}{3}$, I expect to find a perfect pair of functions after 3 attemps. So the expected total running time is $O(n)$ as it takes $O(n)$ to check whether a pair of hash functions is perfect.

The resulting description is small: it consists of 4 numbers: the a's and b's of the two 2-universal hash functions.