Lingfu Zhang February 16, 2015

6.046 Problem 2-2

Collaborators: *none*

(a) The input of this problem are a key k and two B-trees T_1 and T_2 with same minimum degree parameter t and same height $h_1 = h_2$. All keys in T_1 are strictly smaller than k while all keys in T_2 are strictly larger than k. The out put should be another B-tree with minimum degree parameter t and contains exactly all keys in T_1 and T_2 plus k

Let the root node of T_1 and T_2 be r_1 and r_2, respectively. Then we create a node r, whose keys are keys in r_1, k, and keys in r_2, in that order. The children of r are children of r_1 and children of r_2, in their previous order, and the children of r_2 come after the children of r_1. Now if $r.n \leq 2t-1$, return r; otherwise, create nodes r_1', r_2' and r', where the keys in r_1' are the first half in r:

$$\{r.key_1, ..., r.key_{[\frac{r.n-1}{2}]}\}$$

and the keys in r_2' are

$$\{r.key_{[\frac{r.n+3}{2}]}, ..., r.key_{[r.n]}\}$$

The children of r_1' are $r.c_1$, ... , r_1' is $r.c_{[\frac{r.n+1}{2}]}$; the children of r_2' are $r.c_{[\frac{r.n+3}{2}]}$, ... , $r.c_{r.n}$. Let r' has only one key $r.key_{[\frac{r.n+1}{2}]}$, and $r'.c_1 = r_1'$ and $r'.c_2 = r_2'$. Then return r'.

The tree starting from r we create in this algorithm contains exactly all keys in T_1, T_2, and k, and as all its nodes except for the root node are from either T_1 or T_2, the number of keys they contain is between $t-1$ and $2t-1$. As T_1 and T_2 has same height, r leads a B-tree except for that the $r.n$ might exceed $2t-1$. If $r.n \leq 2t-1$, r itself is qualified. However, if $r.n > 2t-1$, as both $r_1.n \leq 2t-1$ and $r_2.n \leq 2t-1$, we have that $r.n \leq 4t-1$. Thus both $r_1'.n$ and $r_2'.n$ are $\leq 2t-1$ and $> t-1$. As r' contains exactly all keys in r, and all other nodes in r' are from T_1 or T_2, we might say that r' is a qualified tree.

Making r, r_1', r_2' and r' all takes $O(t)$ time, and this time is considered constant.

(b) The input of this problem are a key k and two B-trees T_1 and T_2 with same minimum degree parameter t and have heights h_1 and h_2, where $h_1 = h_2 + 1$. All keys in T_1 are strictly smaller than k while all keys in T_2 are strictly larger than k. The out put should be another B-tree with minimum degree parameter t and contains exactly all keys in T_1 and T_2 plus k

Let the roots of T_1 and T_2 are r_1 and r_2 respectively. Let the last child of r_1 be $r_l.lc$. Then $r_1.lc$ has the same height as r_2. We do the same algorithm in (a) on $r_1.lc$ and r_2, and assume the result is r_g. If the tree leading by r_g has height $h_1 - 1$, this indicates $r_g.n = r_1.lc + r_2 > t - 1$, and $r_g.n < 2t - 1$. We can replace $r_1.lc$ with r_g, and return r_1.
On the other hand, if the tree leading by r_g has height h_1, this indicates that $r_g.n = 1$. Add a key, which equals $r_g.key_1$, to r, and that key should be the largest key of r. Let the two nodes before and after that key be the two child of r_g (thus $r_1.lc$ is overwritten). Now return r_1 if $r_1.n \leq 2t - 1$; otherwise, do the same split that we did on r in (a), and return the tree we get.

According to (a), the r_g we get here is a qualified B-tree with all keys from the tree under $r_1.lc$ and T_2. If it has height $h_1 - 1$, or h_2, replacing $r_1.lc$ with r_g simply adds all keys in T_2. As $r_g.n$ is within the range, r_1 is what we need. If r_g has height h_1, the process we take also deletes $r_1.lc$ from the tree leading by r_1 and adds r_g, and the number of keys in all nodes are between $t - 1$ and $2t - 1$, except for r_1, which might have $2t$ keys. This can be handled using the same approach we dealt with r in (a).

Getting r_g requires constant time, according to (a). Merging r_g with r_1 under both conditions also takes constant time, as we need at most add one key to r_1 and redirect two children. The possible splitting of r_1 still takes constant time, according to (a). Thus the total time is constant.

(c) The insertion takes an augmented B-tree and a key k not in the tree as input, returns another augmented B-tree that contains all keys in the input B-tree and k; the deletion takes an augmented B-tree and a key k that is in the tree as input, returns a B-tree that contains all keys in the input tree except for k. All B-trees here have the same minimum degree t.

Both algorithms are similiar to the algorithms for non augmented B-tree. We let $x.h$ be the height of the subtree below x. For the insertion, the "B-TREE-SPLIT-CHILD(x, i)" in CLRS 18.2 should be modified. We need to let $z.h = y.h$ after creating z, as the subtree below y will be split to two below y and z, respectively. Besides, for "B-TREE-INSERT(T, i)", we should let $s.h = r.h + 1$ when creating s.

As at most one s will be created during one insertion, and split at most h times, the time added is $O(h)$, and the total time is $O(h)$.

For deletion,