Biased 2-b trees
(Bent, Sleator, Tarjan 1980)

1

Goal

Keep sorted lists subject to the following operations:

find(x,L)

insert(x,L)

delete(x,L)

catenate(L1,L2) : Assumes that all items in L2 are greater than all items in L1. (2-way catenate)

catenate(L1,i,L2) : Assumes that all items in L2 are greater than i and all items in L1 are less than i. (3-way catenate)

split(x,L) : returns two or three lists. One with all item less than x, the other with all items greater than x, and the third with x if its in L.

2

In addition

- Items have weights: Item i has weight w_i
- Let $W = \Sigma w_i$ be the total weight of the items
- Want the search to heavy items to be faster
- If $p_i = w_i/W$ represents the access frequency to item i then the average access time is

$$\Sigma \; p_i \, d_i$$

where d_i is the depth of item i

3

There is a lower bound

$$\Sigma \; p_i \, d_i \;\geq\; \Sigma \; p_i \log_b (1/p_i)$$

for every tree with maximum degree b

So we will be looking for trees for which $d_i = O(\log (W/w_i))$

In particular if all weights are equal the regular search trees which we have studied, will do the job.

4

Biased 2-b trees definition

Internal nodes have degree between 2 and b. We also need an additional property:

Define the rank of a node x in a 2-b tree recursively as follows.

If x is a leaf containing item i then

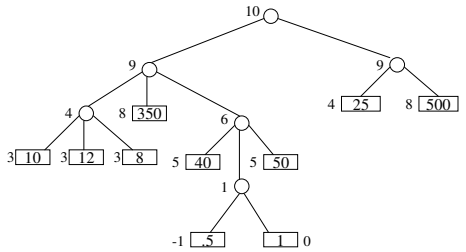
$r(x) = \lfloor \log_2 w_i \rfloor$

If x is an internal node

$r(x) = 1 + \max \{ r(y) \mid y \text{ is a child of } x \}$

5

Biased 2-3 tree (example)



6

Biased 2-b trees definition (cont)

Call x major if r(x) = r(p(x)) - 1

Otherwise x is minor

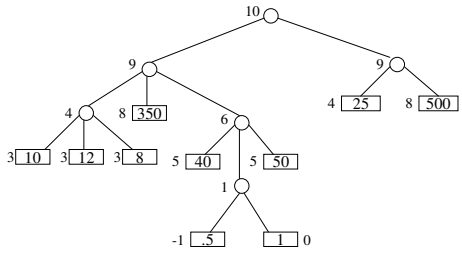
Here is the additional property:

Local bias: Any neighboring sibling of a minor node is a major leaf.

In case all weights are the same this implies that all leaves should be at the same level and we get regular 2-b trees.

7

Biased 2-3 trees example revisited



8

Are the access times ok ?

Define the size of a node x in a 2-b tree recursively as follows.

If x is a leaf containing item i $s(x) = w_i$

If x is an internal node $s(x) = \Sigma_{\text{y is a child of x}}\ s(y)$

Lemma: For any node x, $2^{r(x)-1} \le s(x)$,

For a leaf x, $2^{r(x)} \le s(x) < 2^{r(x)+1}$

==> if x is a leaf of depth d then $d < \log(W/ w_i) + 2$

proof. $D \le r(root) - r(x) < \log (s(r)) + 1 - (\log(s(x)) - 1)$

9

Are the access times ok ? (cont.)

Lemma: For any node x, $2^{r(x)-1} \le s(x)$,

For a leaf x, $2^{r(x)} \le s(x) < 2^{r(x)+1}$

proof. By induction on r(x).

If x is a leaf the definition $r(x) = \lfloor \log_2 s(x) \rfloor$ implies that $2^{r(x)} \le s(x) < 2^{r(x)+1}$

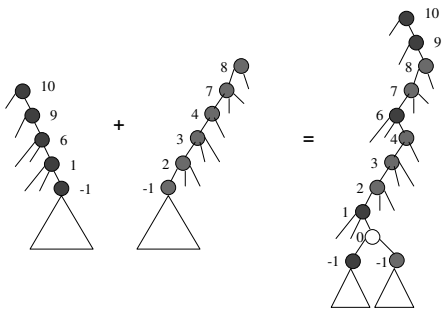
If x is an internal node with a minor child then x has a major child which is a leaf, say y.
So $2^{r(x)-1} = 2^{r(y)} \le s(y) < s(x)$

If x is an internal node with no minor child then it has at least two major children y and z:

$2^{r(x)-1} = 2^{r(y)-1} + 2^{s(z)-1} \le s(y) + s(z) \le s(x)$

10

Concatenation (example)



11

Catenation (definition)

Traverse the right path of the tree rooted at x and the left path of the tree rooted at y concurrently. Go down one step from the node of higher rank. Stop either when they are both equal or the node of higher rank is a leaf.

Merge the traversed paths ordering nodes by rank:

Case 1: If the rank of the rank-largest node of the last two nodes is one smaller than the rank of the smallest-rank node w above this pair then stick the last two nodes as children of w.

Merge the paths by rank.

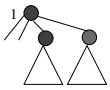
Split w if necessary and continue splitting as long as a major node splits (the nodes resulting from the split have the same rank).

When a minor node splits add a new node which is a parent of the two node resulting from the split and stop. Otherwise, you stop when the root splits

12

Catenation (definition)

Why does a node split into two nodes of the same rank ?



One of the new guys is major (both except for the first step) and one of the old guys is major.

Catenation (definition)

Case 2: If the rank of the rank-largest node of the last two nodes is smaller by at least 2 than the rank of the smallest-rank node w above this pair then stick the last two nodes as children of a new node g. Stick g underneath the smallest parent of the last two node.

Merge the paths by rank.

Catenation (proof of correctness)

Follows from the following observations:

Obs1: Before splitting every minor node stands where a minor node used to stand before in one of the trees.

Obs2: Splitting preserves local bias.

Catenation (worst case analysis)

Worst case bound:

$O(\max\{r(x),r(y)\} - \max\{r(u),r(v)\}) = O(\log(W/(w_- + w+))$

x and y are the two roots

u is the rightmost leaf descendant of x and
v is the leftmost leaf descendant of y

$w_- = s(u)$, $w_+ = s(v)$,
W is the total weight of both trees.

In particular if y is leaf and x is the root of a big tree of weight W then this bound is O(W/s(y))

Catenation (amortized analysis)

amortized bound: $O(|r(x) - r(y)|)$

Proof:

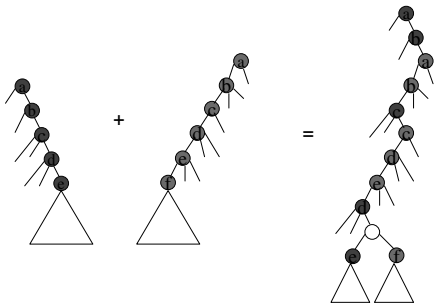
We want the potential to decrease by one for every node of rank smaller than r(y) that we traverse.

Potential (def):

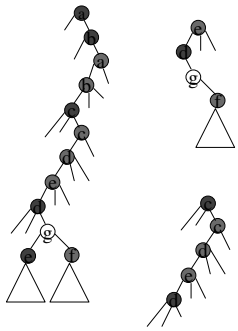
every (minor) node x has r(p(x)) - r(x) - 1 credits.

Φ = total number of credits.

Catenation (amortized analysis)



Catenation (amortized analysis)

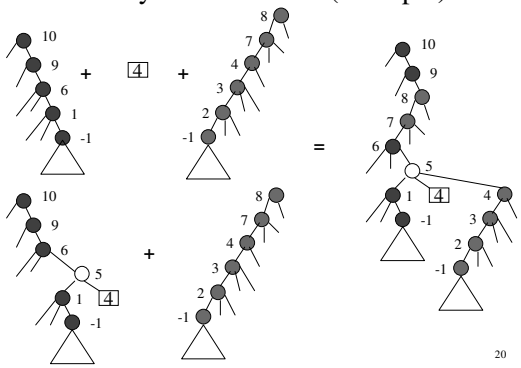


f had r(e) - r(f) - 1 credits.
g needs r(d) - r(g) - 1
which is smaller by at least 2,
in general it would be smaller by at least 1 + the number of blue guys

d had r(c) - r(d) - 1
d needs r(e) - r(d) - 1
# of released credits is at least the number of pink guys

19

3-way concatenation (example)



20

3-way concatenation

Do two succesive 2-way catenations.

Analysis:

Amortized: O(max{r(x), r(y), r(z)} - min{r(x), r(y), r(z)})

worst-case: O(max{r(x), r(y), r(z)} - r(y))

21

2-way split

Similar to what we did for regular search trees.

Suppose we split at a leaf y which is in the tree. We go up from y towards the root x and accumulate a left tree and a right tree by succesive 2-way catenations

Analysis: To split a tree with root x at a leaf y.

amortized: O(r(x) - r(y)) = O(log(W/s(y))

22

3-way split

Splitting at an item i which is not in the tree.

Let i- be the largest item in the tree which is smaller than i

Let i+ be the smallest item in the tree which is bigger than i

Let y be the lowest common ancestor of i- and i+

The initial left tree is formed from the children of y containing item less than i. The initial right tree is formed from the children of y containing items bigger than i.

Analysis: To split a tree with root x at an item i not in the tree

amortized: O(r(x) - r(y)) = O(log(W/(s(i-) + s(i+)))

23

Other operations

Define delete, insert, and weight change in a straightforward way in terms of catenate and split.

24

Extensions

There are many variants.

Binary variants.

Variants that has good bounds for all operations on the worst case

25