Splay trees
(Sleator, Tarjan 1983)

1

Goal

Support the same operations as previous search trees.

2

Highlights

- binary
- simple
- good amortized property
- very elegant
- interesting open conjectures -- further and deeper understanding of this data structure is still due

3

Main idea

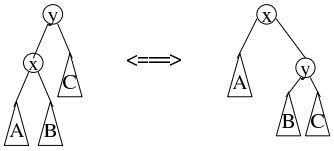
- Try to arrange so frequently used items are near the root

- We shall assume that there is an item in every node including internal nodes. We can change this assumption so that items are at the leaves.

4

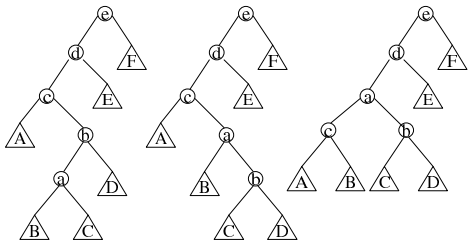
First attempt

Move the accessed item to the root by doing rotations



5

Move to root (example)



6

Move to root (analysis)

There are arbitrary long access sequences such that the time per access is O(n) !

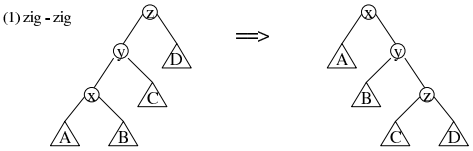
7

Splaying

Does rotations bottom up on the access path, but rotations are done in pairs in a way that depends on the structure of the path.

A splay step:

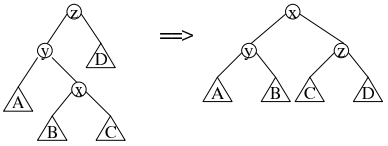
(1) zig - zig



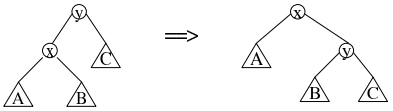
8

Splaying (cont)

(2) zig - zag

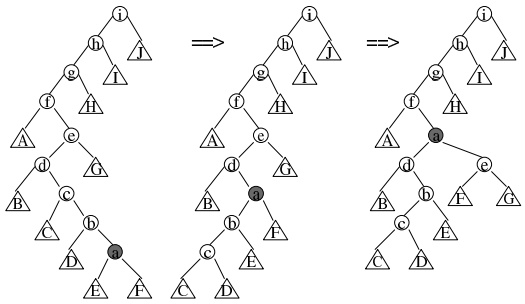


(3) zig



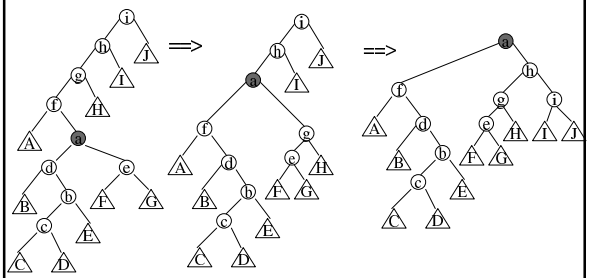
9

Splaying (example)



10

Splaying (example cont)



11

Splaying (analysis)

Assume each item i has a positive weight w(i) which is arbitrary but fixed.

Define the size s(x) of a node x in the tree as the sum of the weights of the items in its subtree.

The rank of x: $r(x) = \log_2(s(x))$

Measure the splay time by the number of rotations

12

Access lemma

The amortized time to splay a node x in a tree with root t is at most $3(r(t) - r(x)) + 1 = O(\log(s(t)/s(x)))$

Potential used: The sum of the ranks of the nodes.

This has many consequences:

Balance theorem

Balance Theorem: Accessing m items in an n node splay tree takes $O((m+n) \log n)$

Proof.

Assign weight of 1/n to each item.

The total weight is then W=1.

To splay at any item takes $3\log(n) +1$ amortized time

the total potential drop is at most $n \log(n)$ □

More consequences after the proof.

Proof of the access lemma

The amortized time to splay a node x in a tree with root t is at most $3(r(t) - r(x)) + 1 = O(\log(s(t)/s(x)))$

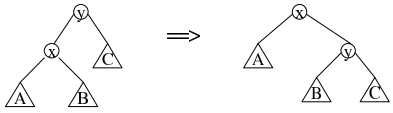
proof. Consider a splay step.

Let s and s', r and r' denote the size and the rank function just before and just after the step, respectively. We show that the amortized time of a zig step is at most $3(r'(x) - r(x)) + 1$, and that the amortized time of a zig-zig or a zig-zag step is at most $3(r'(x)-r(x))$

The lemma then follows by summing up the cost of all splay steps

Proof of the access lemma (cont)

(3) zig



amortized time(zig) = $1 + \Delta\Phi =$

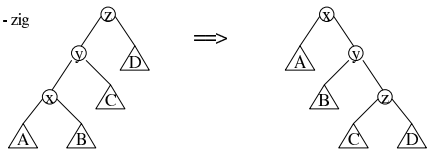
$1 + r'(x) + r'(y) - r(x) - r(y) \leq$

$1 + r'(x) - r(x) \leq$

$1 + 3(r'(x) - r(x))$

Proof of the access lemma (cont)

(1) zig - zig



amortized time(zig) = $1 + \Delta\Phi =$

$2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) =$

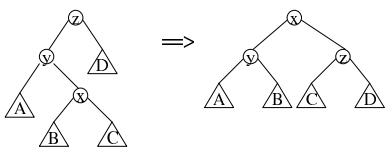
$2 + r'(y) + r'(z) - r(x) - r(y) \leq$

$2 + r'(x) + r'(z) - 2r(x) \leq$

$2r'(x) - r(x) - r'(z) + r'(x) + r'(z) - 2r(x) = 3(r'(x) - r(x))$

Proof of the access lemma (cont)

(2) zig - zag



Similar. (do at home)

Static optimality theorem

For any item i let q(i) be the total number of time i is accessed

Static optimality theorem: If every item is accessed at least once then the total access time is $O(m + \sum_{i=1}^{n} q(i) \log (m/q(i)))$

Splay trees are as good as biased search trees designed knowing q(1),q(2),...,q(n).

Recall that biased search tree have optimal average access time up to a constant factor.

Static optimality theorem (proof)

Static optimality theorem: If every item is accessed at least once then the total access time is $O(m + \sum_{i=1}^{n} q(i) \log (m/q(i)))$

Proof. Assign weight of q(i)/m to item i.

Then W=1.

Amortized time to splay at i is $3\log(m/q(i)) + 1$

Maximum potential drop over the sequence is

$\sum_{i=1}^{n} \log(W) - \log (q(i)/m)$ □

Application: Data Compression via Splay Trees

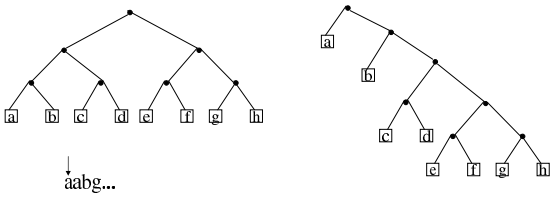
Suppose we want to compress text over some alphabet Σ

Prepare a binary tree containing the items of Σ at its leaves.

To encode a symbol x:

- Traverse the path from the root to x spitting 0 when you go left and 1 when you go right.
- Splay at the parent of x and use the new tree to encode the next symbol

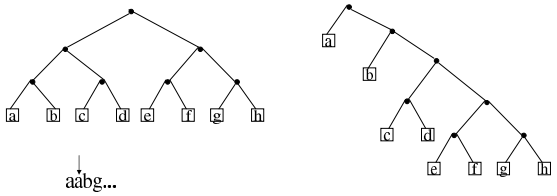
Compression via splay trees (example)



aabg...

000

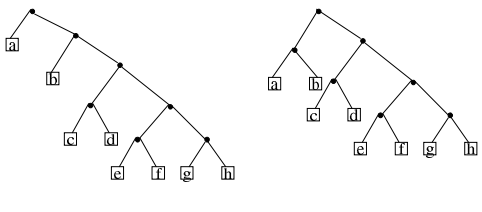
Compression via splay trees (example)



aabg...

0000

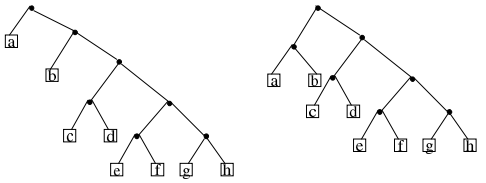
Compression via splay trees (example)



aabg...

000010

Compression via splay trees (example)



aabg...

0000101110

Decoding

Symmetric.

The decoder and the encoder must agree on the initial tree.

Compression via splay trees (analysis)

How compact is this compression ?

Suppose m is the # of characters in the original string

The length of the string we produce is m + (cost of splays)

by the static optimality theorem

$m + O(m + \Sigma\, q(i) \log (m/q(i))) =$
$O(m + \Sigma\, q(i) \log (m/q(i)))$

Recall that the entropy of the sequence $\Sigma\, q(i) \log (m/q(i))$ is a lower bound.

Compression via splay trees (analysis)

In particular the Huffman code of the sequence is at least

$\Sigma\, q(i) \log (m/q(i))$

But to construct it you need to know the frequencies in advance

Here we also produce a code which is alphabetic:

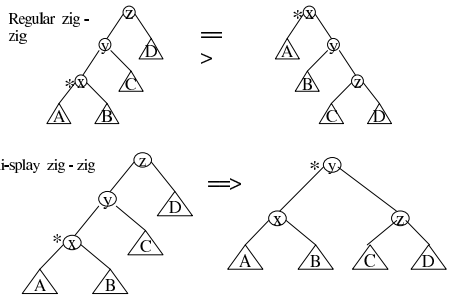
The initial ordering of the characters carries over to their encodings.

Compression via splay trees (variations)

D. Jones (88) showed that this technique could be competitive with dynamic Huffman coding (Vitter 87)

Used a variant of splaying called semi-splaying.

Semi - splaying



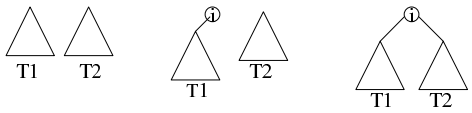
Continue splay at y rather than at x.

Update operations on splay trees

Catenate(T1,T2):

Splay T1 at its largest item, say i.

Attach T2 as the right child of the root.



Amortize time: $3(\log(s(T1)/s(i)) + 1 + \log(\frac{s(T1)+s(T2)}{s(T1)})$

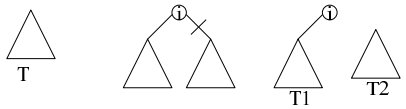
$= 3\log(W/w(i)) + O(1)$

Update operations on splay trees (cont)

split(i,T):

Assume $i \in T$

Splay at i. Return the two trees formed by cutting off the right son of i



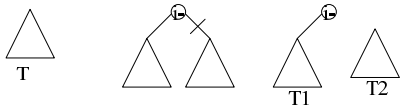
Amortized time = $3\log(W/w(i)) + O(1)$

Update operations on splay trees (cont)

split(i,T):

What if $i \notin T$?

Splay at the successor or predecessor of i (i- or i+). Return the two trees formed by cutting off the right son of i or the left son of i



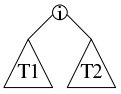
Amortized time = $3\log(W/\min\{w(i-),w(i+)\}) + O(1)$

Update operations on splay trees (cont)

insert(i,T):

Perform split(i,T) ==> T1,T2

Return the tree

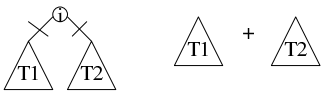


Amortize time: $3\log(\frac{W-w(i)}{\min\{w(i-),w(i+)\}}) + \log(W/w(i)) + O(1)$

Update operations on splay trees (cont)

delete(i,T):

Splay at i and then return the catenation of the left and right subtrees



Amortize time: $3\log(W/w(i)) + 3\log(\frac{W-w(i)}{w(i-)}) + O(1)$

Open problems

Self adjusting form of a,b tree ?

Open problems

Dynamic optimality conjecture:

Consider any sequence of successful accesses on an n-node search tree. Let A be any algorithm that carries out each access by traversing the path from the root to the node containing the accessed item, at the cost of one plus the depth of the node containing the item, and that between accesses perform rotations anywhere in the tree, at a cost of one per rotation. Then the total time to perform all these accesses by splaying is no more than O(n) plus a constant time the cost of algorithm A.

43