Advanced Data Structures
Lecture 8

Lecture by Dr. Shay Mozes
Typeset by Steven Karas

2016-12-29
Last edited 21:02:00 2016-12-29

Agenda So far, we've been looking for setting lower bounds on various static/dynamic optimums in order to prove competition. We will show a lower bound on all binary trees for some sequence of operations, and then show a tree that is $O(\log \log n)$-competitive.

Clarification It's important to note that we are willing to allow potential functions to become negative up to some constant value c.

1 Lower bound on dynamic binary trees

Wilber, SICOMP 89'

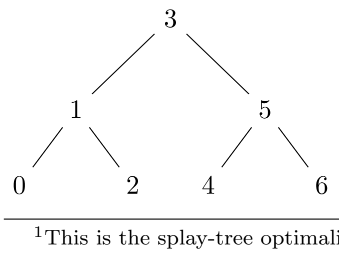
Assumption Only access operations

Assumption Keys are $1, ..., n$

The trivial lower bound for $\sigma = \sigma_1, ..., \sigma_m$ is $\Omega(m)$. Recall that the amortized cost for splay trees is $O(\log n)$.

Define $W(P, \sigma)$ where P is a static tree, and σ is the sequence of operations. We will show that for any σ, P, and for any dynamic tree T the number of operations that T takes for the sequence σ. We want to find the P that maximizes W. If we were able to find for all σ, T a tree P for which splay-trees do $O(W(P, \sigma))$ operations, then we have proven that the splay trees are $O(1)$-competitive to the dynamic optimum.[1]

Example:



[1] This is the splay-tree optimality conjecture, an open problem.

For each node in the tree, mark the last edge that was used to access an element as the "favored" branch. If our sequence of accesses is $0, 4$, then we first mark $3 \to 1, 1 \to 0$, and then unmark $3 \to 1$, and mark $3 \to 5, 5 \to 4$.

If we access the tree from the leaves moving up, we use the bit reversal sequence (add one to each, then reverse the bits).

$$0, 4, 2, 6, 0, ...$$

Assume for a moment we access only the leaves. This sequence maximizes the "swapped" branches for each access. Let $W(P, \sigma)$ be the times we switch the favored branches. For our example P and σ, $W(P, \sigma) = \Theta(n \log n)$. Note that splay-trees always take $O(m \log n)$.

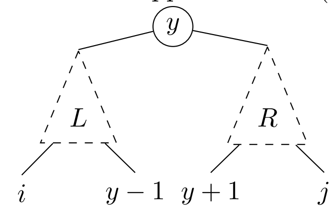
Proof Let σ be a sequence of accesses. Let P be a static tree, and T be a dynamic tree. The number of operations that T must take to fulfill the sequence σ is $\Omega(W(P, \sigma))$

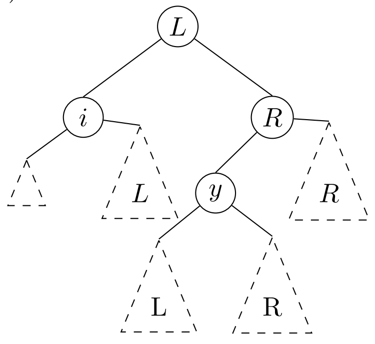
For all $y \in P$, we will define a critical node $z(y) \in T$[2]. We will define this mapping such that:

1. At any point, $z(y) \neq z(y')$ for any $y \neq y'$.
2. If $z(y)$ is the critical node for y at time t, then it holds that T touches $z(y)$ at the latest the second time that y switches it's favored branch after time t.

From properties 1 and 2 it follows that T performs at least $\frac{1}{2}W(P, \sigma)$.

Definition of $z(y)$ Given a static tree P such as this (note that i is the smallest element in the subtree rooted at y, and j is the largest), the dynamic tree T can appear as such (but need not):





Define $z(y)$ be the highest node in T such that the path from the root passes through nodes both from L and from R.

Let $LCA(a, b)$ be the lowest common ancestor of both a and b. The highest node from L is $LCA(i, y)$. The highest node from R is $LCA(y, j)$.

Claim One of $LCA(i, y)$ and $LCA(y, j)$ is an ancestor of the other.

[2]In the original paper, these are called "transition points"

Claim $z(y)$ is the lowest of $LCA(i, y)$ and $LCA(y, j)$.

Claim $z(y) \neq z(y')$ for any $y \neq y'$

Claim If $z(y) = LCA(y, j)$, then all the nodes in R are descendants of $z(y)$. A similar claim for the converse is also made.

Claim $z(y)$ only changes when we touch $z(y)$ (when accessing or rotating T).

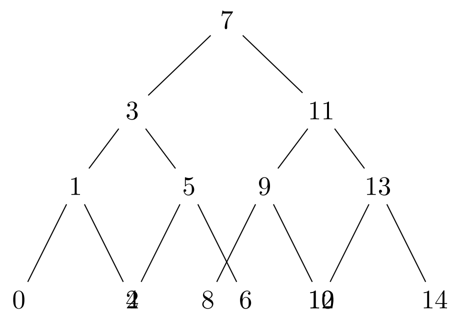
A visual proof of the second property was done on the whiteboard (using colors).

2 Tango trees

Binary search tree that for any sequence σ does $W(P, \sigma) \log \log n$ operations, where P is a balanced static tree.

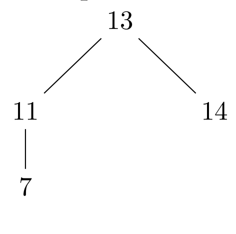
The basic idea is to break up the tree into favored paths.

Example:



We will represent each "favored" path using a BST. Each BST has $O(\log n)$ nodes. For each node x in each BST, we keep a pointer to the root of the BST which contains the unfavored branch of x. The depth of x in P. The maximum depth in P of all the nodes underneath x in the BST.

Example:



access(x)**:** Start from the BST that contains the root of P (the favored path). Search for predecessor and successor of x. If we have not found x in the BST, we continue to the unfavored path of either the predecessor or successor. If we change the favored paths, we need to update the BSTs.

Complexity Searching for the pred/succ: $O(\log \log n)$. Checking which tree x belongs to costs us $O(1)$. Each time we switch BSTs to the unfavored branch, this costs OPT $O(1)$.

Therefore, for a sequence σ of operations it costs us $O(W(P, \sigma) \cdot \lg \lg n)$.

Updating the favored path BSTs We want to cut the previously favored subtree, and link the new (using split/join). These cost us $O(\lg \lg n)$. However, we need to split these by the maximum depth. Mark l and r as the smallest and largest elements in the previously favored branch of x in P. Note that we can split based on some interval. Next week we will show the specifics of this operation.