6.897: Advanced Data Structures Spring 2003

Lecture 8 — Monday, March 10, 2003

Prof. Erik Demaine *Scribe: Lance Anderson and William Kwong*

1 Overview

1.1 Last Lecture

In the last lecture we explored splay trees and how they can improve search time in a tree.

1.2 Today

In this lecture we will see some properties of splay trees and try to learn how fast they really are. We will discuss two properties, the working set property and the dynamic finger property. Then we will see how these are combined to form a unified property. These properties are important because tell us about the speed of any data structure that satisfies them.

2 Properties of Splay Trees

Splay trees perform better by a constant factor. One natural question to ask is how fast splay trees work. First, let us explore two important properties about splay trees: the working-set property and the dynamic finger property. Note that the working set property and dynamic finger property may also be present in data structures other than splay trees.

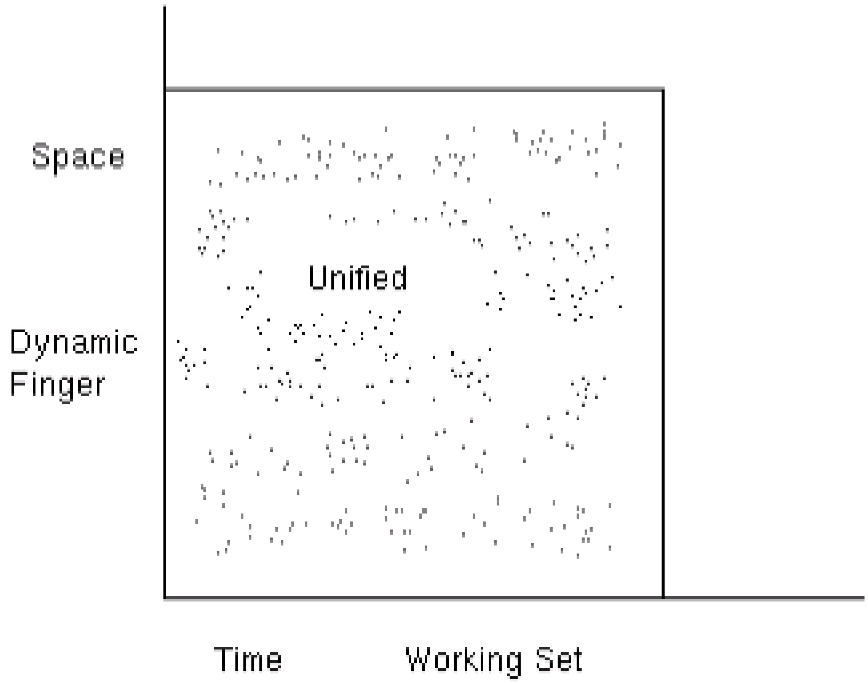
2.1 Working set property

The working state property states that if I have accessed an element recently, assessing that element again will be cheap. The amortized cost to access $x = O(1 + \lg(\text{number of distinct elements accessed since last access to } x))$.

2.2 Dynamic finger property

The dynamic finger property states that if the element I am accessing is close to an element I accessed before, the current cost will be cheap. Amortized cost to access x after $y = O(lg(1+|x-y|)$ where $|x-y|$ is measured in rank.

2.3 Unified property



The example below shows why we care about the area denoted by rectangle, which corresponds to the unified property.

Example:

Request Sequence	Dynamic Finger	Working Set	Unified
$1, 2, \ldots, n, 1, 2, \ldots, n$	$\Theta(m)$	$\Theta(m \lg n)$	$O(m)$
$1, 2, \ldots, n, 1, n, 1, n$	$\Theta(mlgn)$	$\Theta(m)$	$O(m)$
$1, n/2, 2n/2 + 1, 3, \ldots, n/2 - 1, n$	$\Theta(m \lg n)$	$\Theta(m \lg n)$	$O(m)$

Unified Property [Iacono 2001] Suppose request sequence is x1, x2, . Xm, then the amortized cost of accessing xi is $<= lgmin(j <= i)(1+|xi - xj|+$ number of distinct elements accessed between xj & $xi)$

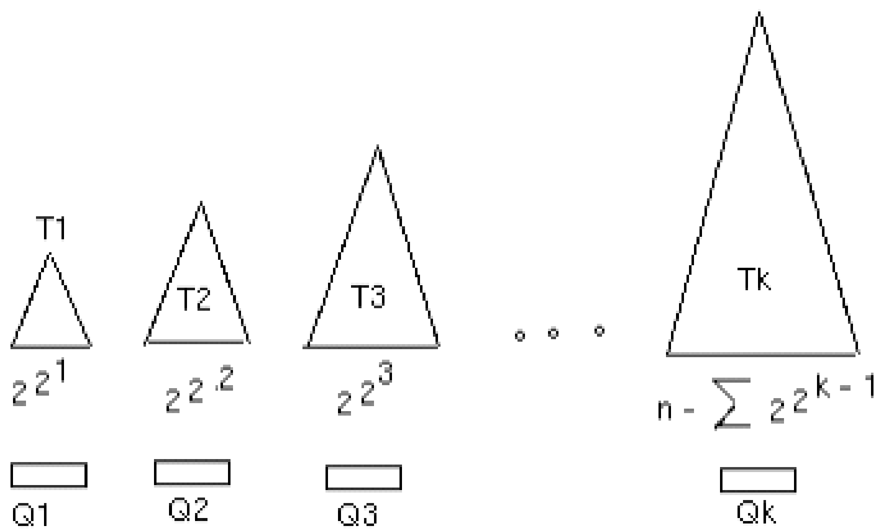
Note that the formula above includes measures for both spatial distance and temporal distance. The unified property is strictly stronger than both the working set property and the dynamic finger property. Note that anything static cannot satisfy the unified property because of the lack of the dynamic finger property. The best performance by static data structures is the entropy bound.

Open Problems 1. Do Splay Trees satisfy the unified property? i.e. Is the unified structure an upper bound for splay trees? (The conjecture is yes.) There has been research on splay tree matching the dynamic optimal. LEMMA: [Fredman, unpublished] there's a sequence where splay trees are faster than the unified bound. 2. Does any dynamic search tree have unified prop?

THEOREM: The unified structure has the unified property. 3. Building a simpler and more practical structure. Currently there are 21 pointers per node. 4. Making the unified structure dynamic.

Our goal is to build a unified structure with the unified property. To get started, we will build a data structure with the working-set property. Note that splay trees exhibit the working-set property, but they are difficult to manipulate.

3 Working Set Structure - A multi-level cache



Picture of "lg lg n" balanced binary search trees of exponentially increasing size:

We will use Qi to denote the ith queue and Ti to denote the ith binary search tree.

The following steps are involved when searching for x:

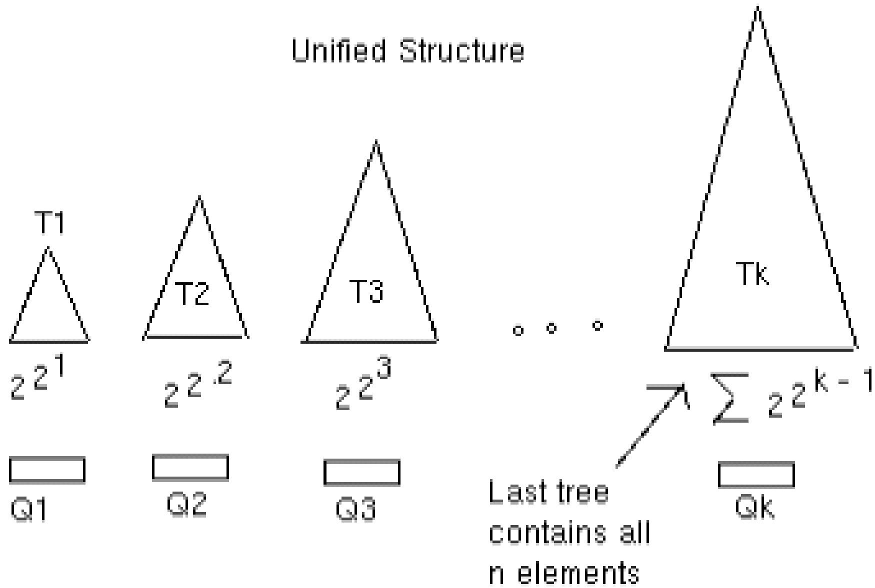
1. Look in T1, T2,. until x is found
2. The unsuccessful cost is: SUM $i = (kto1)$ of $(O(lg(2^{2^i}))) = O(lgn)$ (in handout)
3. Delete x from TL and QL.
4. Insert x into T1 and enqueue x into Q1
5. "Shift from 1 to L"

Note that steps (iii) and (iv) are like the move to front operation.

Now we will proceed to measure the cost of searching for x. Note that the number of distinct elements accessed since x was last accessed is called the working set number, w. The number of binary search trees touched, L, is lg lg w. The time cost is: SUM from (i = 1 to $lglgw$) of: $(lg2^{2^i}) = O(2^l glgw) = O(lgw) + O(1)$

Deletes and inserts are also possible using shift.

4 Unified structure



When comparing the unified structure with the working set structure, we see that the last tree in the unified structure has all n elements. In contrast, the last tree in the working set structure contains only leftover elements. In addition, queues in the unified structure will not store all elements in Ti. Each Qi only stores "childless" elements in Ti.

Every node in Ti has a parent in Ti+1. i.e. every node in a tree is pointing to another node in the next tree. A node is only stored once in a given tree. The number of childless elements in Ti is at least 2 to the $2^I - (2^{2^{i-1}})$. Thus Qi stores ends up storing most elements. From the data structure, we can see that element x in Ti is close to its parent; it is within distance $2^{2^{i+1}}$ of parent (x) in $Ti+1$.

For any random query, search (x), we can find in $O(2^i)$ time whether there's an element in Ti within $2*2^{2^i}$ of x. When searching for x in Ti and seeing where it fits, we don't know the rank of x in keyspace. This is the comparison model and we do not know how close adjacent elements.

Search(x) involves the following steps:

- Find where x fits in Ti. - Binary search in T1T2T3 for elements within $2*2$ to the (2^i) of x. - Suppose TL has y within 2 to the (2^L) of x. Then proceed with as follows: Add x -¿ TL with same parent as y Add x -¿ T(L-1) with same parent as x in TL

Add x -¿ T1 with parent=x in T2

We assume each add operation takes $O(2^L)$. The result is that x will be in a continuous group of trees.

If x is within $2*2^{2^i}$ of y in Ti, then x is within $2*2^{2^{i+1}}$ of parent (y) in $Ti+1$. The proof states that if x is within $2*2^{2^i}$ of y, it is within $(2*2^{2^i} + 2^{2^{i+1}}) <= 2*2^{2^{i+1}}$ of parent y. In particular, right after x is accessed, x is within 2^{2^i} of parent $(i-1)$ in Ti. In fact, any element within $2^{2^i} - 2*(2^{2^{i-1}}$

of x is also within $2 * 22^i$ of $Pi - 1$ in Ti. $Pi - 1$ survives for $>= (2^{2^i} - 2^{2^{i-1}})/2$ searches after x is accessed.

If we search for element y within time $O(2^{2^i} - 2^{2^{i-1}})/2$ and within space $O(2^{2^i} - 2^{2^{i-1}})$ of x, then y is within $2 * 2^{2^i}$ of parent(i-1) in Ti. Since L¡i, the cost for this search is simply $O(lg(2^{2^i}))$ and this structure is unified.

Summary of Case 1: when x is within $2^{2^{i+1}}$ of parent (y) - insert x -¿ Ti - enqueue x -¿ Qi - Link x ¡-¿ parent (y) - Ti has to preserve its size, hence remove element at end of queue: dequeue Z from Qi. - - Delete z from Ti. - Unlink z ¡-¿ parent(z) - Maybe enqueue parent (z) -¿ Qi+1

Summary of Case 2: when x is not within $2^{2^{i+1}}$ of parent (y)

5 References

1. Nearly Optimal Expected-Case Planar Point Location - Sunil Arya Theocharis (2000)
2. Entropy-Preserving Cuttings and Space-Efficient Planar Point Location - Sunil Arya Theocharis (2001)
3. A Simple Entropy-Based Algorithm for Planar Point Location - Arya, Malamatos, Mount (2001)
4. On the Exact Worst Case Query Complexity of Planar Point Location - Seidel, Adamy (1998)
5. Improvements to Graph Coloring Register Allocation - Briggs, D.Cooper, Torczon (1994)
6. Overcoming Schematic Discrepancies in Interoperable Databases - Saltor Castellanos (1993)
7. Unification as Constraint Satisfaction in Structuredw Connectionist Networks - Andreas Stolcke (1989)
8. Line Search in Potential Reduction Algorithms for Linear.. - Yinyu Ye (1989)
9. On the Dynamic Finger Conjecture for Splay Trees. Part II: The Proof - Cole
10. Dynamic point location in fat hyperrectangles with integer.. - Iacono, Langerman
11. Optimal Planar Point Location - John Iacono