Advanced Data Structures Jan-Apr 2012

Lecture 3 January 20, 2012

Lecturer: Venkatesh Raman *Scribe: S M Meesum*

1 Overview

In the last lecture we studied Move-to-Front (MTF) Heuristic for a list and its competitive ratio. We also introduced Binary Search Trees (BST) and *optimal BSTs*. In today's lecture, we will be analysing *Splay Trees* and see that they perform as well as an *optimal static BST* without maintaining extra information for balancing the tree.

We also discuss the scenario when we have the freedom to begin the search anywhere instead of always starting from the root node.

2 Statically Optimal Search

Given a sequence of length m and having n distinct elements. We look at the elements one by one. We want to implement following operations:

- *insert (i)* – If i has not been seen so far in the sequence, insert it in the tree.
- *access(i)* – If i exists in the tree, return a pointer to it.

Operation of *insert(i)* is $O(n^2)$ atmost. We can ignore insertion and w.l.o.g assume we have a tree with all the keys initially and we are performing a sequence of *access(i)* operations on it.

2.1 Information Entropy and Search

Given a set of keys $S = [1 \dots n]$, and frequency of access p_i for $i \in S$. The information entropy H is,

$$H = -\sum_i p_i ln(p_i)$$

The expected cost of search for an element is $O(H)$, which for a uniform distribution is $O(log(n))$. Huffman coding acheives this bound, a *static optimum BST* acheives it to a constant factor.

2.1.1 Statically Optimum Tree

If we have are given full access sequence, then we can construct a tree before we start searches. Such a statically optimal tree is contructed using dynamic programming. Optimal BST minimizes the following function,

$$Cost(T) = \sum_i p_i depth_T(i)$$

We have discussed this in the previous lecture. See Knuth's paper [1] for details. The tree is not changed once contructed. The optimal cost per access satisfies following inequality,

$$H - log[H] - log[e] + 1 \leq \frac{opt - cost}{access} \leq H + 3$$

For a proof of above see [10].

2.2 Motivation for using Splay Trees

We can construct a *statically optimum tree* if we have frequencies in advance. Suppose we do not have frequencies in advance. Can we try some heuristic like Move-to-Root, analogous to MTR for lists? Suppose we have an access sequence of $1, 2, \ldots, n, 1, 2, \ldots, n, \ldots$. If we use single rotations to bring accessed element to root then our tree transforms as shown in the figure,

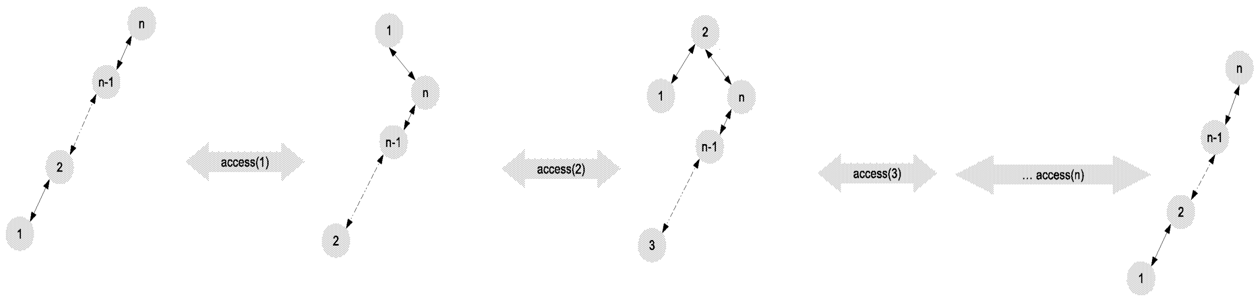

Figure 1: *MTR using Single Rotations*

Taking $O(n)$ time per operation, which is a bad performance compared to online BSTs ($O(log(n))$ per access). A single rotation can cause the tree to grow to linear height. Instead, if we do double rotations, by considering parent and grandparent of a node for rotation, we get a structure which is $O(1)$ competitive with *statically optimum trees.*

These are called *Splay Trees* and were introduced by Sleator and Tarjan [2]. They are easier to implement without extra balancing information for balancing in BSTs. The statically optimal BSTs have the expected search time of $O(log(n))$ for all inputs, while *Splay Trees* being dynamic in nature perform better if there is some order in the input. For the access sequence described above *Splay Trees* take $O(1)$ time per access.

3 Splay Trees

Suppose x is accessed in a *Splay Tree*. Let p be its parent. When p is not a root node we use ZigZag (Figure: 2) and ZigZig (Figure: 3) rotations. When p is the root node we use Zig (Figure: 4) rotation. Zig rotation is not used if the node is at an even distance from the root node.

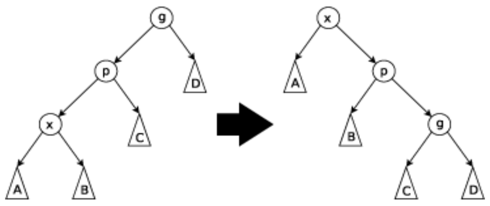

Figure 2: *Zig-Zig Rotation of a Splay Tree*

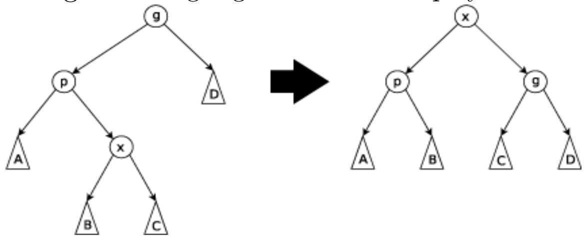

Figure 3: *Zig-Zag Rotation of a Splay Tree*

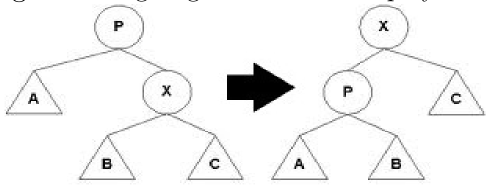

Figure 4: *Zig Rotation of a Splay Tree*

Rotations are performed till x reaches the root node. For insertion, search is started at root node and wherever the search ends we insert the new element, and then bring it to the root.

Cost Model: The cost incurred per access is the number of rotations to move the element to root, which is proportional to its depth.

3.1 Amortized Analysis of Splay Trees

Let T be a *Splay Tree*. Suppose at t^{th} step x is accessed. T_i is the tree configuation in intermediate steps, obtained by ZigZag, ZigZig and finally Zig rotation (which places x at the root node). Define rank *r(x)* of node x to be,

$$r(x) = log\left(\sum_{y \in T[x]} S(y)\right)$$

Where $T[x]$ is the subtree rooted at x. Choosing $S(y) = 1$, rank is the *log* of subtree size. The potential is the sum of the ranks of all the nodes.

$$\phi(t) = \sum_{x \in T} r(i)$$
$$amcost(i) = actcost(i) + \phi(T_i) - \phi(T_{i-1})$$

Lets consider the ZigZig (Figure: 2) rotation. $r(x)$ and $r'(x)$ refer to rank of node x before and after rotation respectively.

$$\begin{aligned} amcost(i) =& 2 + r'(x) + r'(p) + r'(g) - r(x) - r(p) - r(g) \\ amcost(i) =& 2 + r'(p) + r'(g) - r(p) - r(x) \\ amcost(i) \leq& 2 + r'(x) + r'(g) - 2r(x) \end{aligned}$$

In above inequality, we used, $r'(x) = r(g)$, $r(p) \geq r(x)$ and $r'(p) \leq r'(x)$. The last sum is atmost $3(r'(x) - r(x))$ ie. we need to prove $r(x) + r'(g) - 2r'(x) \leq -2$. For this, use the result that $log(a) + log(a)$, for $a, b \geq 0$ is maximized at -2 when $a = b = \frac{1}{2}$. Denote the size of subtree rooted at x by $s(x)$.

$$\begin{aligned} r(x) + r'(g) - 2r'(x) =& \\ log(s(x)/s'(x)) + log(s'(z)/s'(x)) \leq& -2 \end{aligned}$$

For above we used, $s(x) + s'(g) \leq s'(x)$. Thus, we finally get,

$$amcost(i) \leq 3(r'(x) - r(x))$$

Using similar reasoning, cost for ZigZag is found to be atmost $2(r'(x) - r(x))$.

The Zig rotation is performed as a last rotation to bring x to root if it does not have a grandparent.

$$amcost_{Zig} \leq 1 + r'(x) - r(x)$$

Adding together all costs we get,

$$\begin{aligned} Amcost(x) \leq& 3(r_{final}(x) - r_{initial}(x)) + 1 \\ Amcost(x) \leq& 3log(n) + 1 \end{aligned}$$

In the final step x is at the root node (so subtree size is n), while $r_{initial}(x) \geq 0$.

3.1.1 Amortized cost in terms of frequencies

Suppose the frequency of access for each element x is $f(x)$. Note that, $\sum_{x \in T} f(x) = m$. We use the same form of rank and potential and choose $S(y) = f(y)$. Thus, rank becomes log of sum of frequencies of elements in subtree rooted at x,

$$\begin{aligned} r(x) :=& log\left(\sum_{y \in T[x]} f(y)\right) \\ \phi(T) =& \sum_{x \in T} r(x) \\ Amcost(x) \leq& 3(r_{final}(x) - r_{initial}(x)) + 1 \end{aligned}$$

For above inequality the previously used analysis works. Substituting values of ranks,

$$\begin{aligned} r_{final}(x) =& log(m) \\ r_{initial}(x) \geq& f(x) \end{aligned}$$

We get,

$$\begin{aligned} Amcost(x) \leq& log\left(\frac{m}{f(x)}\right) + 1 \\ Amcost(x) \leq& - log(p_i) + 1 \end{aligned}$$

Summing over the access sequence we see that *Splay Trees* acheive *static optimality.*

3.2 Static Finger

Throughout the above discussion, we always assumed the search for an element begins from the root node. The element where we begin search changes everytime a different element arrives at root node. Suppose we were given a pointer to an element and start the search from it everytime a query arrives. We call such a pointer *static finger.* Suppose we were given pointer to a fixed element x. Searching y should now take $O(log(d(x - y)))$.

3.2.1 Amortized Analysis for Static finger

For a fixed finger f. Choose $S(y) = \frac{1}{1+(y-f)^2}$. Use the same form of potential and rank function. The amortized complexity for searching element i is $O(log|i - f|)$.

4 Some Theorems

4.1 Working Set Theorem

Suppose x is searched in i^{th} operation. Let $w_i(x)$ denote the number of distinct elements searched since the last search to x. For example, if the sequence queried is $7, 9, 5, 6, 4, 3, 2, 4, 3, 5$, then $w_{10}(5)$ is 4. The *Working Set Theorem* states that,

Theorem 1. *The amortized cost of accessing an element x in i^{th} step is $log(w_i(x))$.*

Intuitively this means that rotations do not throw away recently accessed elements far from root.

4.2 Dynamic Finger Theorem

The Dynamic Finger Theorem was first conjectured in [2] and finally proven by Cole et al.[3], [4],

Theorem 2. *The amortized cost of accessing the* i^{th} *element* x_i *is* $O(\log[1 + |x_i - x_{i-1}|])$.

4.3 Scanning Theorem

The Scanning Theorem, demonstrated by Tarjan [7], states the following:

Theorem 3. *If elements of a splay tree are accessed sequentially in order, then the total running time is linear, regardless of the initial structure of the splay tree.*

5 Conjectures

5.0.1 Dynamic Optimality Conjecture

In our analysis we compared performance of a *Splay Tree* against a static offline optimal tree. If we remove the restriction of being static, and allow the tree to adjust itself after access requests, we may get a better algorithm. Call the most efficient such algorithm OPT. *Dynamic Optimality Conjecture* [2] states that,

Conjecture 4. *The run time of splay trees is within a constant factor of OPT for any access sequence.*

If true, it would imply that *Splay Trees* are as good as any other dynamic algorithm specifically designed for an access sequence. They can be easily shown to be at least $O(\log n)$-competitive dynamically. A result on this problem has reduced the competitive gap from $O(\log n)$ to $O(\log \log n)$ [6]. We would discuss this result in the next lecture.

Currently it has not been proved that any known search tree is dynamically optimal.

5.1 Unified Bounds Conjecture

In the lecture, we have shown several properties of splay trees, Lets see how they apply on specific access sequences.

- 1,2, . . . , n, 1,2, . . . , n . . .
  Working Set $O(log(n))$. Dynamic Finger $O(1)$.
- 1, n, 1, n, . . .
  Working Set $O(1)$. Dynamic Finger $O(log(n))$.
- 1, n/2, 2, n/2 + 1, 3, n/2 + 2 . . .
  Working Set $O(log(n))$. Dynamic Finger $O(log(n))$.

The third sequence must take time $O(1)$ per access if dynamic optimality holds. Why ?

It was observed in [9] that working-set bound is stronger than other bounds.

- The Working set theorem implies the static finger theorem, which in turn implies the static optimality theorem, in any data structure.
- The Dynamic finger theorem implies the static finger theorem.

In addition, Iacono [8] has proposed the following *unified conjecture*:

Conjecture 5. *The amortized cost to access element x_i is upper bounded by:*

$$O(\min_y \log[w_i(y) + |x_i - y| + 2])$$

Intuitively, this suggests that element x_i is cheap to access if there exists some different element y that is both spatially (in terms of the structure of the tree) and temporally (in terms of access time) close to it. The unified conjecture, if true, would imply both the working set and dynamic finger theorems. It is not known whether the unified conjecture holds for a BST, but Badoiu and Demaine [5] have shown that it can be done on a pointer machine.

5.2 Traversal Conjecture

Conjecture 6. *Supposing we have two splay trees, each with the same elements. Take a preorder traversal of one of the splay trees to obtain an access sequence. The conjecture is that the cost of accessing elements in this access sequence in order in the other splay tree is $O(n)$ [2].*

This conjecture generalises the *Scanning Theorem.* Suppose that T is a splay tree with keys drawn from $[1..n]$. Suppose T' is another tree on the same keys. This conjecture becomes the *Scanning Theorem* when T' has 1 at root node and all other keys are right children of their parents. Preorder traversal of T' is then simply the keys in sequential order. This conjecture says that for any T' the cost of accessing is $O(n)$.

References

[1] D. Knuth, *Optimum binary search trees*, Acta Informatica 1, p. 14-25, 1971.

[2] D. Sleator and R. Tarjan, *Self-Adjusting Binary Search Trees*, Journal of the ACM, 32, 652-686, 1985.

[3] R. Cole, B. Mishra, J. Schmidt, and A. Siegel, *On the dynamic finger conjecture for splay trees. Part I: Splay sorting* $\log n$*-block sequences*, SIAM Journal of Computing, 30(1), 1-43, 2000.

[4] R. Cole, *On the dynamic finger conjecture for splay trees. Part II: The proof*, SIAM Journal of Computing, 30(1), 44-85, 2000.

[5] M. Badoiu and E. Demaine, *A Simplified, Dynamic Unified Structure*, Latin America Theoretical Informatics, 466-473, 2004.

[6] E. Demaine, D. Harmon, J. Iacono, M. Patrascu, *Dynamic Optimality–Almost*, Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004), p. 484-490, October 17-19, 2004.

[7] R. Tarjan, *Sequential access in splay trees takes linear time*, Combinatorica 5(5), 367-378, 1985.

[8] J. Iacono, *Alternatives to Splay Trees with* $o(\log n)$ *worst case access times*, Symposium on Discrete Algorithms, 516-522, 2001.

[9] John Iacono, *Distribution Sensitive Data Structures.* PhD thesis, Rutgers, The State University of New Jersey, New Brunswick, New Jersey, 2001.

[10] J. Rissanen, *Bounds for Weight Balanced Trees*, IBM J. Res. Develop, 1973.