6.897: Advanced Data Structures	Spring 2003

Lecture 4 — February 24, 2003
Fusion Trees

Prof. Erik Demaine — *Scribe: Ben Leong*

1 Overview & Review/Background

In Lecture 2, we introduced the Transdichotomous RAM model of computation. In the Transdichotomous RAM model, the memory is composed of words of length $\lg u$ bits each, where u is the size of the universe $\mathcal{U}$. Intuitively, it should take only one word to name an item. The manipulation of a word in this model takes $O(1)$ time for operations like addition, subtraction, multiplication, division, AND, OR, XOR, and left/right shift.

In Lecture 3, we discussed how the fixed-universe successor problem can be solved with y-fast trees [Wil84] in the Transdichotomous RAM model with insert, delete, succesor, and predecessor operations taking only $O(\lg \lg u)$ time.

In this lecture, we will see how we can approach the same problem using *fusion trees* [FW93]. Fusion trees will perform particularly well when u is very large with respect to the number n of elements. More specifically, our goal is to perform the operations insert, delete, successor and predecessor in time better than the $O(\lg n)$ obtained by the usual balanced binary serach tree.

2 Top-Level Idea

A fusion tree is essentially a B-tree [BM72] with branching factor $B = (\lg n)^{1/5}$. What is the height of the B-tree? If h is the height of the B-tree, then $n = B^h$, so

$$\begin{aligned} n &= ((\lg n)^{1/5})^h \\ \Longrightarrow \quad \lg n &= \frac{h}{5} \lg \lg n \\ \Longrightarrow \quad h &= \Theta\left(\frac{\lg n}{\lg \lg n}\right) \end{aligned}$$

Thus, to obtain $O\left(\frac{\lg n}{\lg \lg n}\right)$ time, our problem is reduced to the following: how do we determine where a given query q fits among the $(\lg n)^{1/5}$ word-length keys at each node in $O(1)$ time. Without some preprocessing, we cannot even afford to look at all the keys ($(\lg n)^{1/5}$ words) in $O(1)$ time! To develop a solution, we begin by assuming a static model, where we do not insert/delete and work out how we can perform a query given a fixed set of keys at a node.

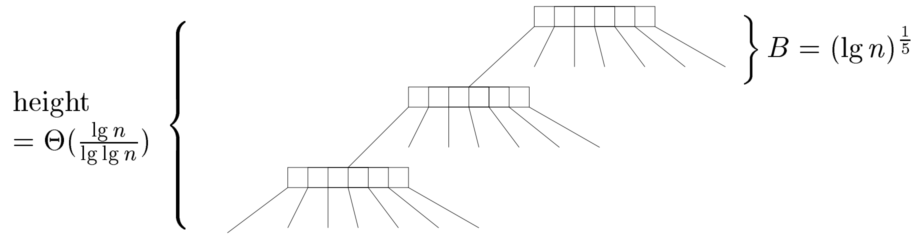
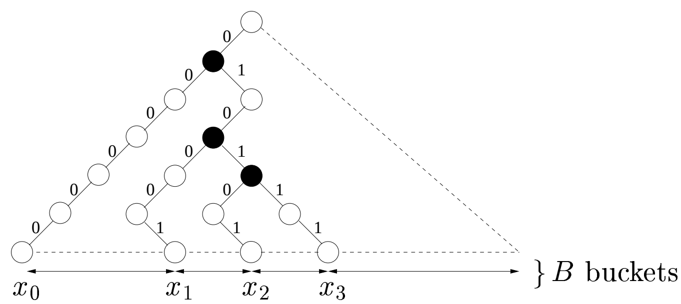


Figure 1: Determining the height of the fusion tree.

3 What matters in a node?

The first idea is to convert all $\Theta(B)$ keys at each node into binary strings and view them as paths in a binary tree. In general, these $\Theta(B)$ keys will partition the address space into $\Theta(B)$ buckets and our task for a given query q is to determine which bucket it fits into. Suppose that the keys in a node are $x_0 < x_1 < \cdots < x_{k-1}$, where $k = \Theta(B)$.



Although the paths may be arbitrarily long, we notice that the branching nodes (shaded in black above) roughly characterize the shape of the binary tree. For k keys, there are exactly $k-1$ branching nodes. Because several branching nodes can occur at a given level, the number of levels that contain branching nodes is less than or equal to $k-1$. This means that we require less than k bits to distinguish between the x_i's. We shall call the bit positions corresponding to the branching levels $b_0, b_1, \cdots, b_{r-1}$, where

$$b_0 < b_1 < \cdots < b_{r-1} \quad \text{and} \quad r \le k - 1 = \Theta(B)$$

Definition 1 (Perfect Sketch). *Perfect Sketch(x) = extract bits $b_0, b_1, \cdots, b_{r-1}$ out of word x, i.e. a r-bit vector where the ith entry is the b_ith bit of x.*

A perfect sketch would allow us to reduce the node representation to k r-bit strings. Because $k = \Theta(B)$ and $r = O(B)$, *sketch*(x) requires $O(B^2) = O((\lg n)^{2/5})$ bits total. This is less than a word.

Problem: Perfect sketch is hard to compute (especially for an arbitrary query q)!

Solution: We compute a slightly less compact form of $sketch(x)$. This approximation of sketch contains the same bits as the perfect sketch in the same order, but the bits are spread out with extra 0's in between (in a pattern that is independent of x, i.e. in consistent positions).

4 Computing $sketch(x)$

We compute $sketch(x)$ using multiplication. This trick uses the word-level parallelism available in the Transdichotomous RAM model. Let us start by trying to multiply by m and see how we should choose m. Suppose $x = \sum_{i=0}^{r-1} x_{b_i} 2^{b_i}$. We can discard all irrelevant bits by computing $\left(x \text{ AND } \sum_{i=0}^{r-1} 2^{b_i}\right)$, leaving just the bits that correspond to the branching levels $b_0, b_1, \cdots, b_{r-1}$. If we multiply x by $m = \sum_{i=0}^{r-1} 2^{m_i}$, we obtain

$$\begin{aligned} \left(x \text{ AND } \sum_{i=0}^{r-1} 2^{b_i}\right) \cdot m &= \left(\sum_{i=0}^{r-1} x_{b_i} 2^{b_i}\right) \cdot \left(\sum_{i=0}^{r-1} 2^{m_i}\right) && (1) \\ &= \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} x_{b_i} 2^{b_i + m_i} && (2) \end{aligned}$$

Now, all we need to do is to choose m so that the bits $b_0, b_1, \ldots, b_{r-1}$ are redistributed in a **compact** way. In particular, we will pick m such that

1. $b_i + m_j$ are distinct $\forall i, j$ (to prevent collisions).
2. $b_i + m_i$ are concentrated in a *small* range[1] (to allow us to deal with it as a single word).
3. $b_0 + m_0 < b_1 + m_1 < \cdots < b_{r-1} + m_{r-1}$, i.e. the order of $b_0, b_1, \ldots, b_{r-1}$ is preserved (so that we do not lose information).

Once we find m, we can compute $sketch(x)$ as

$$sketch(x) = \left(\left(\left(x \text{ AND } \sum_{i=0}^{r-1} 2^{b_i}\right) \cdot m\right) \text{ AND } \sum_{i=0}^{r-1} 2^{b_i + m_i}\right) >> \min_i \{b_i + m_i\}. \qquad (3)$$

Here $\left(\left(\left(x \text{ AND } \sum_{i=0}^{r-1} 2^{b_i}\right) \cdot m\right) \text{ AND } \sum_{i=0}^{r-1} 2^{b_i + m_i}\right)$ extracts all the bits from the word x that correspond to the branching levels $b_0, b_1, \ldots, b_{r-1}$ by masking out all the extraneous bits generated by the multiplication, i.e., all the bits $2^{b_i + m_j}$ where $i \neq j$. We are then left with r bits within a range of r^4 bits. We simply determine the lowest-order bit and right-shift until we remove all the trailing zeros and we are left with the desired r^4-bit word.

Now we return to the issue of picking m, or more precisely, the m_i's.

First: Pick $m'_0 < m'_1 < \cdots < m'_{r-1} < r^3$ such that $b_i + m_j$ are distinct modulo r^3 $\forall\ i, j$. We prove by induction that this is possible.

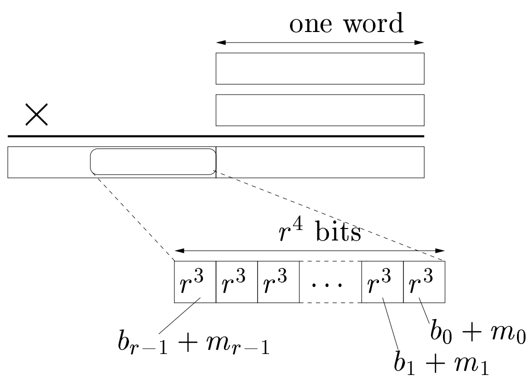
[1] More specifically, within a range of r^4 bits.

Proof: When $r = 1$, we have only one term and clearly the condition is trivially satisfied. Assume that we are able to pick $m'_0 < m'_1 < \cdots < m'_{t-1}$ such that $b_i + m_j$ are distinct modulo r^3. We observe that in order to pick m'_t such that the condition is still satisfied, we only have to avoid all the terms $m_i + b_j - b_k \; \forall \, i, j, k$. Otherwise,

$$\begin{aligned} m'_t &\equiv m_i + b_j - b_k \pmod{r^3} \\ \implies \quad m'_t + b_k &\equiv m_i + b_j \pmod{r^3} \text{ (BAD!)} \end{aligned}$$

To avoid all the terms $m_i + b_j - b_k \; \forall \, i, j, k$, we have to avoid $t \cdot r \cdot r \leq (r-1)r^2$ terms, because $t < r$. Because the number of terms we have to avoid is $(r-1)r^2$, which is less than the size r^3 of the address space, there must be at least one term that we do not have to avoid[2].

Second: We add suitable multiples of r^3 to m'_i to produce m_i so that the condition $b_0 + m_0 < b_1 + m_1 < \cdots < b_{r-1} + m_{r-1}$ is satisfied and all the bits fall just beyond the highest bit of the low half of the product, as shown in the following figure:



Clearly, we can find the appropriate m_i's if we choose the multiples for $m'_0, m'_1, \ldots, m'_{r-1}$ in strictly increasing order. We observe that the resulting sketch has $\leq r^4$ bits.

Definition 2 (Sketch(Node)). *We define the sketch for an entire node as follows:*

$$sketch(node) = 1\,\overbrace{sketch(x_0)}^{\leq r^4 \text{ bits}}\,1\,\overbrace{sketch(x_1)}^{\leq r^4 \text{ bits}} \cdots 1\,\overbrace{sketch(x_{k-1})}^{\leq r^4 \text{ bits}} \tag{4}$$

Essentially, it is the concatenation of the sketches[3] of all the keys in the node, separated by 1's.

Because each $sketch(x_i)$ is $\leq r^4$ bits in length and there are k of these terms, the total length of $sketch(node)$ is $\leq (r^4+1)k$, taking into account the extra 1's that are concatenated between the $sketch(x_i)$'s. Because $r \leq k-1$, $r^4 + 1 < k^4$. Hence, the length of $sketch(node) < k^5 \leq B^5 = \lg n$ bits. This means that $sketch(node)$ will fit in one word!

[2]This is a simple application of the *Pigeon-hole Principle*, where there are less pigeons than there are holes. Clearly, at least one hole must be empty!

[3]Note that the function $sketch(x_i)$ (for all keys and all queries) at a node is a constant function determined only by the keys (which is possible because we are considering only the static case).

5 Finding where a query fits in the sketch world

Once we compute the sketch of a node, we want to be able to use the sketch to allow us to compute efficiently where a query q fits with respect to all the keys at the node. To do so, we first compute $sketch(q)$ using (3) above. Next, we multiply $sketch(q)$ by a number generated by concatenating k repetitions of the (r^4+1)-bit sequence of the form "$00\cdots01$", i.e.,

$replicated\text{-}sketch(q) =$

$$sketch(q) \times (\underbrace{\overbrace{00\cdots01}^{r^4+1\text{ bits}}\overbrace{00\cdots01}^{r^4+1\text{ bits}}\cdots\overbrace{00\cdots01}^{r^4+1\text{ bits}}}_{k\text{ terms}}) = \underbrace{0sketch(q)0sketch(q)\cdots0sketch(q)}_{0sketch(q)\text{ repeated }k\text{ times}} \tag{5}$$

Next, we subtract $replicated\text{-}sketch(q)$ from $sketch(node)$, i.e.,

$$sketch(node) - replicated\text{-}sketch(q) = \underbrace{\overbrace{c_0\cdots\cdots}^{r^4+1\text{ bits}}\overbrace{c_1\cdots\cdots}^{r^4+1\text{ bits}}\cdots\overbrace{c_{k-1}\cdots\cdots}^{r^4+1\text{ bits}}}_{k\text{ terms}} \tag{6}$$

If we AND this expression with $\sum_{i=0}^{k-1} 2^{i(r^4+1)+r^4}$, we obtain

$$\left(\underbrace{\overbrace{c_0\cdots}^{r^4+1\text{ bits}}\overbrace{c_1\cdots}^{r^4+1\text{ bits}}\cdots\overbrace{c_{k-1}\cdots}^{r^4+1\text{ bits}}}_{k\text{ terms}}\right) \text{ AND } \left(\sum_{i=0}^{k-1} 2^{i(r^4+1)+r^4}\right) = \underbrace{\overbrace{c_0 0\cdots0}^{r^4+1\text{ bits}}\overbrace{c_1 0\cdots0}^{r^4+1\text{ bits}}\cdots\overbrace{c_{k-1}0\cdots0}^{r^4+1\text{ bits}}}_{k\text{ terms}} \tag{7}$$

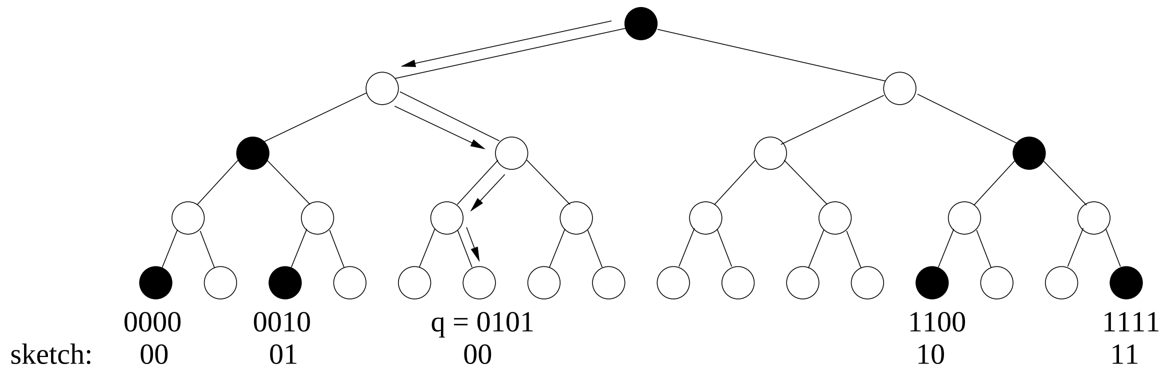
We observe that

$$c_i = \begin{cases} 0 \text{ if } sketch(x_i) < sketch(q) \\ 1 \text{ if } sketch(x_i) \geq sketch(q) \end{cases} \tag{8}$$

and that c_i as a sequence is zero for a while and then all 1's and all that remains is for us to find the crossover point. As discussed in Lecture 3, there are many ways to find the most significant bit of (7). One is to use the $O(1)$-time algorithm of [Bro93]. We cannot afford to precompute a lookup table on half-words, because this would require $\Theta(\sqrt{u}\lg\lg u)$ space, which may be much larger than n. However, we can apply a different sketch function to (7) to be able to use a lookup table. Because we care about only k bits, we can compute the sketch and extract the k bits. More specifically, we can extract all the relevant k bits into a bit string of length $\leq k^4$ bits and perform a lookup on the entire string. Because $k^4 = \Theta(B^4) = \Theta((\lg n)^{4/5})$, the space required for the lookup table is $O(2^{(\lg n)^{4/5}}\lg\lg n) = o(n)$.

6 Knowing where you are in the sketch world is not enough!

Even though we have shown that we can determine where a query fits in the sketch world, we still need something more. Consider the tree with 4 elements and an incoming query "0101" as shown in the following figure:



If we compute $sketch(q)$, we obtain "00" and hence it would seem that q belongs somewhere between "0000" and "0010." Yet, this is clearly not the case.

Problem: Finding where q fits in the sketch world is not sufficient.

Solution: We first find the two neighbors of q in the sketch world, say x_i and x_{i+1}. Next, we find the longest common prefix/lowest common ancestor of the *real elements* (not sketch) of either q and x_i or q and x_{i+1} (whichever is longer/lower). This is equivalent to finding the most significant 1-bit (again!) in the XOR of the 2 elements (which we know how to do from Lecture 2). This bit is where we "fell off" the correct path and identifies the node of divergence. If we fix this first wrong bit and set all remaining bits to either all 0's or 1's, we can find the minimum/maximum in the subtree via another query in the sketch world and our query is complete. □

Next time: In the next lecture, we shall see how we can make fusion trees dynamic and faster.

7 Open Problems (Demaine & Iacono 2003)

A natural question that we can ask is: how can we generalize fusion trees and the van Emde Boas data structure to higher dimensions? Let's start with the 2-dimensional case.

Problem 1: More specifically, how much more efficiently can we solve the *planar point location problem*? In this problem, a plane is divided into cells by a graph. Given a query point, the goal is to find the cell that contains the point. (See Figure 2.) Preprocessing is allowed on the planar graph, but queries are performed online. The goal is to come up with an algorithm that can do better than $O(\lg n)$ time.

Problem 2: A special case of this problem involves an infinite strip in a plane. The strip is divided into trapezoids by line segments. Given a query point, the goal is to find the trapezoid that contains the point. (See Figure 3.) The challenge is to solve this problem in $o(\lg n)$ time.

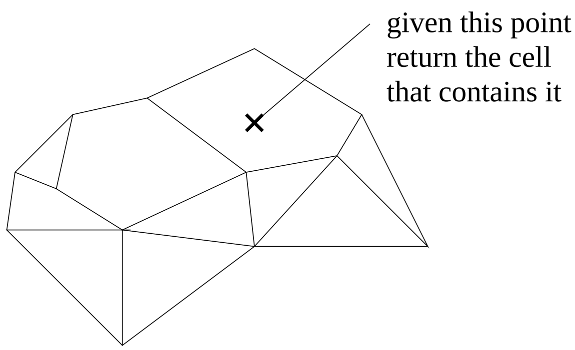


Figure 2: Planar Point Location Problem.

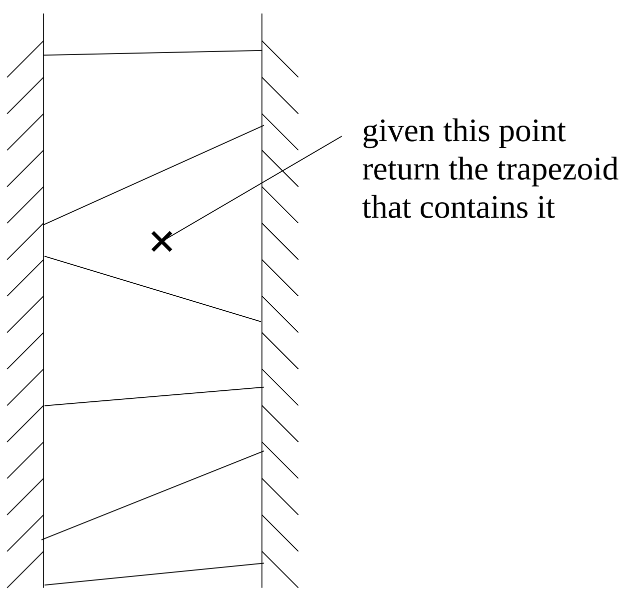


Figure 3: Special Case of Planar Point Location Problem.

References

[BM72] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[Bro93] A. Brodnik. Computation of the least significant set bit. In *Proceedings Electrotechnical and Computer Science Conference*, volume B, pages 7–10, 1993.

[FW93] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[Wil84] D. E. Willard. Log-logarithmic worst-case range queries are possible in space Theta(n). *Information Processing Letters*, 17:81–84, 1984.