Advanced Data Structures Jan-Apr 2012

Lecture 10 — February 17, 2012

Lecturer: Venkatesh Raman *Scribe: Shion Samadder Chaudhury*

1 Overview

In the last lecture we looked at datastructures that efficiently support the actions *predecessor*, *successor* along with the usual *membership*, *insertion* and *deletion* actions.

In this lecture we discuss Fusion trees. Our goal is to perform the above mentioned actions in time better than $O(\log\ n)$. Given a static set S $\subseteq (0, 1, 2, ..., 2^w - 1)$, fusion trees can answer predecessor/successor queries in $O(\log_w n)$. Fusion Tree originates in a paper by Fredman and Willard [1].

2 Main Idea

2.1 Model

In our model, the memory is composed of words each of length $w = \log u$ bits where u is the size of our universe U. Each item we store must fit in a word. The manipulation of a word in this model takes $O(1)$ time for operations like addition, subtraction, multiplication, division, AND, OR, XOR, left/right shift and comparison.

2.2 Description of Fusion tree

A Fusion tree is a B-tree with a branching factor of $k = \Theta(w^{1/5})$. Let h be the height of the B-tree. Then

$$h = log_{w^{1/5}} n = \frac{1}{5} log_w n = \Theta(log_w n)$$

To get an $O(\log_w n)$ solution to the problem we have to find a way to determine where a query fits among the B keys of a node in $O(1)$ time.

3 Nodes of the Fusion Tree

Let us suppose that the keys in a node are $x_0 < x_1 < ... < x_{k-1}$ where $k = \theta(B)$ with each of them a w-bit string. We view each of them as a root-to-leaf path in a binary tree whose edges are labeled 0 or 1. When we get 0 we move to the left and we move to the right when we get 1. This is basically a trie.

A node in this trie is called a *branching node* if it has a non-empty left and right subtree. There are $k - 1$ such nodes. Shown in the following figure.

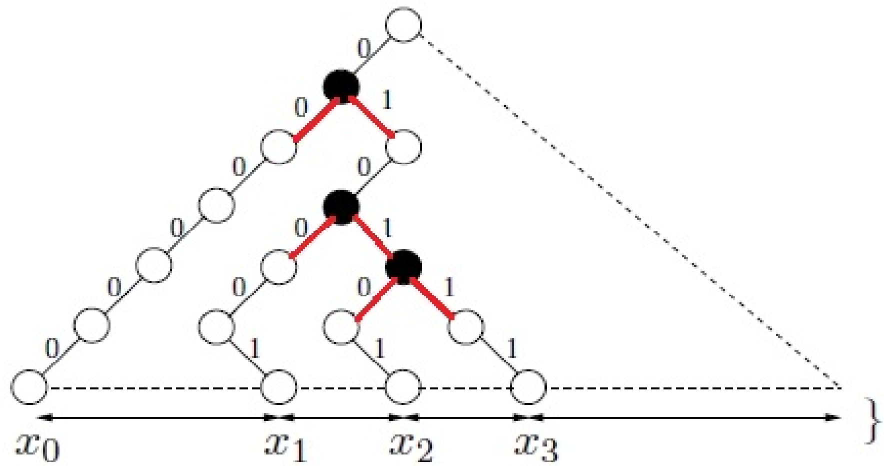

Figure 1: the black nodes are the branching nodes

We mark the horizontal levels in the trie containing the branching nodes. A level in the trie corresponds to a bit position in a w-bit string. Let $b_1 < b_2 < ... < b_r$ the bit positions corresponding to the levels in the trie. These bit positions are called the *important bits* of the node. There are at most $k - 1 = O(w^{1/5})$ *important bits.*

3.1 Sketch

As we mentioned before, we have to find a way to determine where a query fits among the keys of a node in $O(1)$ time. We do this by looking only at the *important bits* of the keys.

Perfect Sketch of a word is the r-bit binary string obtained by extracting the *important bits* from it. ie,

$$sketch(\sum_{i=0}^{w-1} 2^i x_i) = \sum_{i=1}^{r} 2^i x_{b_i}$$

Since $r \leq k - 1 = O(w^{1/5})$ we can concatenate the *sketches* of the k keys of a node to get a bit string of length $O(w^{2/5})$, which will fit in one word.

But *perfect sketch* is hard to compute under the operations of our integer word RAM model.

So we compute an *approximate sketch* of size $O(w^{4/5})$ in constant time. Using this we can fit the sketches of $O(w^{1/5})$ keys into a word. This *approximate sketch* contains the same bits as the *perfect sketch* in the same order but are separated by extra 0's in between.

We want $sketch(\sum_{i=0}^{w-1} 2^i x_i) = \sum_{i=1}^{r} 2^{c_i} x_{b_i}$ where $c_1 < c_2 < ... < c_r < O(w^{4/5})$ are computed from the b_i's.

We compute this as follows: Let $x = \sum_{i=0}^{w-1} 2^i x_i$.

1. We extract only the b_i bits to get $\sum_{i=1}^{r} 2^{b_i} x_{b_i}$.

2. Find $m_1 < m_2 < ... < m_r < r^3$ so that $b_i + m_j$ are distinct modulo r^3.

3. Add correct multiples of r^3 to m_i's to get m'_i so that the condition : $w \leq m'_1 + b_1 < m'_2 + b_2 < ... < m'_r + b_r < w + O(r^4)$ holds.

4. Take $c_i = m'_i + b_i - w$.

So we multiply $\sum_{i=1}^{r} 2^{b_i} x_{b_i}$ by $m = \sum_{i=1}^{r} 2^{m'_i}$ to get $\sum_{j=1}^{r}\sum_{i=1}^{r} 2^{m'_j + b_i} x_{b_i}$. The powers of 2 in this expression are distinct. Hence if we mask to consider only bits of the form $m'_i + b_i$, we are left with $\sum_{i=1}^{r} 2^{m'_i + b_i} x_{b_i}$. Finally, dump the low-order word to get the required.

The validity of the above computation is confirmed by the following theorem :

Theorem : Given the b_i's as above, then

a. there exist constants $m_1 < m_2 < ... < m_r < r^3$ such that $b_i + m_j$ are distinct modulo r^3.

b. suitable multiples of r^3 can be added to m_i's to get m'_i so that the condition : $w \leq m'_1 + b_1 < m'_2 + b_2 < ... < m'_r + b_r < w + O(r^4)$ holds.

Proof: (*part a*) (By induction) When $r = 1$, we have only one term and the condition is trivially satisfied. Let us assume that we have $m_1 < ... < m_t$ such that $b_i + m_j$ are distinct modulo r^3. We observe that m_{t+1} must be different from the terms $(m_i + b_j - b_k)$ modulo r^3 for all i, j, k. So m_{t+1} must be different from $t.r^2 \leq (r-1)r^2$ terms. But $(r-1)r^2$ is less than the size of the address space. Hence there must be at least one term which is feasible.

Proof: (*part b*) We want to get m'_i by adding to m_i a correct multiple of r^3 such that $w + r^3(i-1) \leq m'_i + b_i < w + r^3 i$. Then the order $w \leq m'_1 + b_1 < m'_2 + b_2 < ... < m'_r + b_r < w + O(r^4)$ will be satisfied. So we define $m'_i := m_i + r^3 i +$ (the greatest multiple of $r^3 \leq (w - b_i)$) (or in other words $(w - b_i)$ rounded down to a multiple of r^3).

4 Predecessor and Successor

On a query q, we compute sketch(q) and compare it with every key simultaneously in parallel. We do this in the following way :

1. Pack the sketches of the keys together with a 1 bit on the left side of each, that is,

$$1 sketch(x_1) 1 sketch(x_2) 1 ... 1 sketch(x_k)$$

.

2. Given $sketch(q)$, we compute

$$0 sketch(q) 0 sketch(q) 0 ... 0 sketch(q)$$

- (repeated k times) - this is $sketch(q)(1 + 2^r + 2^{2r} + ... + 2^{kr})$. Let us call it $repeated - sketch(q)$. Therefore,

$$repeated - sketch(q) = sketch(q) * (00...0100...01...00...01)$$

3. If $sketch(q) \geq sketch(x_i)$ the difference between these two words has a 0 in the $r(i-1)$th place from the left. Otherwise it is 1. ie.,

$$sketch(node) - repeated - sketch(q) = (c_0 ....c_1 ......c_i ....c_{(k-1)} ......)$$

where $c_i = 0$ if $sketch(q) \geq sketch(x_i)$, 1 otherwise.

(Note) *Sketch* preserves order of the keys we built it on. $sketch(x_1) < sketch(x_2) < ... < sketch(x_k)$ - so if $sketch(x_j) < sketch(q) < sketch(x_j + 1)$, bits $0, r, ..., r(j-1)$ from the left are 0 and bits $rj, ..., r(k-1)$ are 1. We want to have the value of j, the index of q's "sketch predecessor." We mask out the "junk" bits with an AND. Then rj is the most significant bit of the comparison word. ie., we compute

$(c_0 ....c_1 ......c_i ....c_{(k-1)} ......)\ AND\ (\Sigma_{i=0}^{k-1} 2^{i(r^4+1)+r^4}) = (c_0 0.....0 c_1 0.....0.....0 c_{(k-1)} 0.....0)$

The MSB is easily computable in AC^0. So j is easy to compute.

(Note) Sketch does not preserve order on all words except for those that branch off from one another at the branching positions we selected (i.e., the keys). Hence x_j and $x_j + 1$ need not be necessarily related to the predecessor or successor of q.

5. Let us suppose that q diverges from its predecessor lower than the point of divergence with the successor. Then, one of x_j or $x_j + 1$ must have a common prefix with q of the same length as the common prefix between q and its predecessor. This is because the common prefix remains identical through sketch, and the sketch predecessor can only deviate from the real predecessor below where q deviates from the real predecessor. We can find the length of the common prefix in $O(1)$ by taking bitwise XOR and finding the MSB.

6. Again, let us assume that the predecessor deviates below. If C is the common prefix, the predecessor is of the form $C0A$, and q has the form $C1B$ for some A and B. We now construct the word $q' = C011...1$ and, as above and calculate x_i such that $sketch(x_i) < sketch(q_0) < sketch(x_i + 1)$ in $O(1)$. This element is q's predecessor among the keys of the node.

7. The sketch predecessor of q' is q's predecessor : Since C is the longest common prefix between q and any x_i, we know that no x_i begins with the string $C1$. Hence, the predecessor of q must begin with $C0$, and it must also be the predecessor of $q' = C011...1$. Now, the predecessor of q' is the same as its sketch predecessor: this follows because all important bits in q' after C are 1, and thus q' remains the maximum in the subtree beginning with C, even after sketching. Thus, the maximum element x_i beginning with C is actually the predecessor of q.

8. Now proceed with the fusion tree query by recursing down the corresponding branch.

5 References

[1] 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.

[2] A. Anderson, P. B. Miltersen, M. Thorup. Fusion trees can be implemented with AC^0 instructions only. *Theor. Comp. Sc.*, 215(1-2):337-344, 1999