6.851: Advanced Data Structures Spring 2007

Lecture 13 — April 2, 2007

Prof. Erik Demaine *Scribe: Hooyoung Chung*

1 Introduction

The topic of this lecture is the fusion tree, another data structure to solve the predecessor/successor problem. Given a static set $S \subseteq \{0, 1, 2, \ldots, 2^w - 1\}$, fusion trees can answer predecessor/successor queries in $O(\log_w n)$.

Essentially, fusion trees are B-trees with a branching factor of $k = \Theta(w^{1/5})$. Tree height is $\log_{w^{1/5}} n = \frac{1}{5} \log_w n = \Theta(\log_w n)$, so we can achieve the desired $O(\log_w n)$ query time if we can solve the predecessor problem in $O(1)$ at each node. Doing this requires ingenuity—we can examine at most $O(w)$ bits in $O(1)$ time, whereas each of our $\Theta(w^{1/5})$ keys is w bits. We will accomplish $O(1)$ per node by looking at only important bits of the keys, and by using parallelism to our advantage.

2 Fusion Tree Nodes

For a node in the structure, consider the keys $x_1 < x_2 < \cdots < x_k$. Each of these is a w-bit string. We can also view it as a root-to-leaf path in a binary tree whose child edges are labeled with 0s and 1s: on encountering a 0 we move to the left child, on 1 we move to the right. (This is basically a trie; we don't need the end marker \$ because the strings are uniformly length w.) A *branching node* is a node in the trie neither of whose subtrees is empty; there are exactly $k - 1$ such nodes. If we mark the horizontal levels of the trie containing branching nodes, there are at most $k-1 = O(w^{1/5})$ of these. Note that a level of the trie corresponds to a bit position in a w-bit string. Call these bit positions $b_1 < b_2 < \cdots < b_r$ the node's important bits.

The *sketch* of a word is the r-bit binary string given by extracting the important bits from it, i.e., $\textsc{sketch}(\sum_{i=0}^{w-1} 2^i x_i) = \sum_{i=1}^{r} 2^i x_{b_i}$. Sketches are brief enough to be very useful to us: since $r \leq k - 1 = O(w^{1/5})$, we can concatenate the sketches of a node's k keys and the result is a bit string of length $O(w^{2/5})$, which will fit in one word. Then, on a query q, we can compute $\textsc{sketch}(q)$ and compare it with every key simultaneously in parallel. The way to do this is to pack the sketches of the keys together with a 1 bit of padding on the left side of each, that is, $\texttt{1}\textsc{sketch}(x_1)\texttt{1}\textsc{sketch}(x_2)\texttt{1}\cdots\texttt{1}\textsc{sketch}(x_k)$. Given $\textsc{sketch}(q)$, we compute $\texttt{0}\textsc{sketch}(q)\texttt{0}\textsc{sketch}(q)\texttt{0}\cdots\texttt{0}\textsc{sketch}(q)$ (repeated k times)—this is just $\textsc{sketch}(q) \times (1 + 2^r + 2^{2r} + \cdots + 2^{kr})$. The difference of these two words has a 0 in the $r(i-1)$th place from the left if $\textsc{sketch}(q) \geq \textsc{sketch}(x_i)$ and a 1 otherwise. This takes 1 operation.

Note that SKETCH preserves order of the keys we built it on—$\textsc{sketch}(x_1) < \textsc{sketch}(x_2) < \cdots < \textsc{sketch}(x_k)$—so if $\textsc{sketch}(x_j) < \textsc{sketch}(q) < \textsc{sketch}(x_{j+1})$, bits 0, r, …, $r(j-1)$ from the left are 0 and rj, …, $r(k-1)$ are 1. We would like to have the value of j, the index of q's "sketch predecessor." Once we mask out the "junk" bits with a single AND, rj is the most significant bit of the comparison word. The MSB is easily computable in AC^0, as well as being a fairly standard

operation on physical microprocessors (e.g. Pentium). Surprisingly, it has been shown ([3]) that MSB is computable with a very complicated (but constant time) series of operations in the word RAM model.

So j is easy to compute. Note that we are not done: x_j and x_{j+1} are not necessarily related to the predecessor or successor of q, as SKETCH does not preserve order on all words—just those that branch off from one another at the branching positions we selected (i.e., the keys). However, they give some information about the predecessor or successor of q. Assume 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. Again, assume it is the predecessor that deviates below. Now the question is how to find the actual predecessor based on the known common prefix. If C is the common prefix, the predecessor is of the form $C\texttt{0}A$, and q has the form $C\texttt{1}B$ for some A and B. We now construct the word $q' = C\texttt{011}\ldots\texttt{1}$ and, as above, calculate x_i such that $\text{SKETCH}(x_i) < \text{SKETCH}(q') < \text{SKETCH}(x_{i+1})$ in $O(1)$. This element is q's predecessor among the node's keys. We now proceed with the fusion tree query by recursing down the corresponding branch.

How are we sure that 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 $C\texttt{1}$. Hence, the predecessor of q must begin with $C\texttt{0}$, and it must also be the predecessor of $q' = C\texttt{011}\ldots\texttt{1}$. 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.

3 Perfect and Approximate Sketches

The sketch function described above is perfect: we pull out precisely the important bits of x. Unfortunately, it is impractical ($\omega(1)$) to compute, at least under the operations of our integer word RAM model. Luckily, we can compute an *approximate sketch* of size $O(w^{4/5})$ in constant time; this is enough to fit the sketches of $O(w^{1/5})$ keys packed into a word. (It turns out that, as PERFECT-SKETCH is in AC^0—see [2]—we can use $k = O(w^{1/2})$ keys on each node in the AC^0 RAM model; this translates to a constant factor in overall query time.)

Like PERFECT-SKETCH, an approximate sketch has the bits b_1, b_2, …, b_r in order, but they may be separated by zeros. Formally, we want $\text{SKETCH}(\sum_{i=0}^{w-1} x_i 2^i) = \sum_{i=1}^{r} x_{b_i} 2^{c_i}$, where $c_1 < c_2 < \cdots < c_r < O(w^{4/5})$ are precomputed from the b_i in polynomial time. We show how to define this function and how to compute it in $O(1)$ with a ***clever use of multiplication***:

1. Construct $m_1, m_2, \ldots, m_r$ so that each of $b_i + m_j$ are distinct modulo r^3. This can be done iteratively: if we have already picked $m_1, \ldots, m_t$ so that there are no conflicts, it is enough to pick an m_{t+1} that is not congruent to any $m_i + b_j - b_k$ modulo r^3 for $1 \le i \le t$ and $1 \le j, k \le r$. Since there are fewer than r^3 numbers to avoid, there must be some value of m_{t+1} that works.

2. Let m'_i equal m_i plus the correct multiple of r^3 to make $w + r^3(i-1) \le m'_i + b_i < w + r^3 i$. These values are in the right order: $w \le m'_1 + b_1 < m'_2 + b_2 < \ldots < m'_r + b_r < w + O(r^4)$.

3. Take $c_i = m'_i + b_i - w$. The algorithm to compute SKETCH(x) is as follows: given $x = \sum_{i=0}^{w-1} x_i 2^i$, we mask to leave only the b_i bits, ending up with $\sum_{i=1}^{r} x_{b_i} 2^{b_i}$.

   Next we ***multiply*** this by $m = \sum_{i=1}^{r} 2^{m'_i}$ to get $\sum_{j=1}^{r} \sum_{i=1}^{r} x_{b_i} 2^{m'_j + b_i}$. From step 2, 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 $\sum_{i=1}^{r} x_{b_i} 2^{m'_i + b_i}$. Finally, dump the low-order word to get the desired function.

4 Dynamic Predecessor/Successor

The amount of work necessary to build a fusion tree node ($w^{O(1)}$) is too great to support fast updates. We can offset this problem using the technique of *indirection* from last lecture. Say that rebuilding a node takes w^c; then we can put the elements into buckets of size $\Theta(w^c)$, within which we maintain BSTs. A query or update thus requires amortized $O(\lg_w n)$, plus $O(\lg w^c) = O(\lg w)$ to search within the representative BST.

An alternative construction, which achieves the current best known dynamic set performance, uses exponential trees for $O(\log_w n + \lg \lg n)$ per operation—see [1].

References

[1] A. Andersson and M. Thorup. Tight(er) worst-case bounds on dynamic searching and priority queues. STOC 2000: 335-342.

[2] Arne Andersson, Peter Bro Miltersen, Mikkel Thorup. Fusion trees can be implemented with AC^0 instructions only. *Theor. Comp. Sc.*, 215(1-2):337-344, 1999.

[3] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424-436, 1993.