Today: Fusion trees
- sketch & why it's enough
- approximate sketch via multiplication
- parallel comparison
- most significant set bit 1 year after ‘cold fusion’ debacle

Fusion trees: [Fredman & Willard - STOC 1990, JCSS 1993]
- store $n \cdot w$-bit integers - here, statically
- $O(\log w n)$ time for predecessor/successor
- $O(n)$ space
- word RAM $\Rightarrow$ predecessor $\leq \min \{ \log w n, \log w^3 \}$

- $\text{AC}^0$ RAM version [Andersson, Miltersen, Thorup - TCS 1999] $\Rightarrow$ ops. are constant-depth (unbounded fan) circuits $\Rightarrow$ no multiplication
- dynamic version via exponential trees: $O(\log w n + \log \log w n)$ deterministic updates [Andersson & Thorup - JACM 2007]
- dynamic version via hashing: [Raman - ESA 1996] $O(\log w n)$ expected updates

- OPEN: $O(\log w n)$ w.h.p. updates?
**Idea:** B-tree with branching factor $\Theta(w^{1/5})$

$\Rightarrow$ height $= \Theta(\log_w n)$

$= \Theta(\log n / \log w)$

- search must visit a node in $O(1)$ time
- not enough time to read the node ($w^{1/5}$ w-bit words) to figure out which child

**Fusion-tree node:**

- store $k = O(w^{1/5})$ keys $x_0 < x_1 < \cdots < x_{k-1}$
- $O(1)$ time for predecessor/successor
- $kO(1)$ preprocessing
Distinguishing $k = O(w^{1/5})$ keys:
- View keys $x_0, x_1, \ldots, x_{k-1}$ as binary strings (0/1)
  i.e. root-to-leaf paths in
  height-$w$ binary tree (left/right)
- $\Rightarrow$ $k-1$ branching nodes
- $\leq k-1$ levels
  containing branching nodes
  i.e. bits where $x_0, x_1, \ldots, x_{k-1}$ first differ
  (first distinct prefix)
- Call these important bits $b_0 < b_1 < \cdots < b_{r-1}$,
  $r < k = O(w^{1/5})$

(perfect) sketch($x$) = extract bits $b_0, b_1, \ldots, b_{r-1}$ from $x$
  i.e. $r$-bit vector whose $i$th bit = $b_i$th bit of word $x$
  $\Rightarrow$ sketch($x_0$)$<$$\cdots <$ sketch($x_{k-1}$)

& can pack (fuse) into one word: $k \cdot r = O(w^{2/5})$ bits
  - computable in $O(1)$ time as AC$^0$ operation
    [Andersson, Miltersen, Thorup - TCS 1999]
  - We’ll see a cool way to compute approximate
    sketch using multiplication & standard ops.

Node search: for query $q$, compare sketch($q$)
  in parallel to sketch($x_0$), ..., sketch($x_{k-1}$)
  - again AC$^0$ operation on $O(1)$ words
    & we’ll see a nice way with standard ops.
  $\Rightarrow$ find where sketch($q$) fits among sketch($x_0$)$<$$\cdots <$ sketch($x_{k-1}$)
  - Want where $q$ fits among $x_0 < \cdots < x_{k-1}$
**Desketchifying:**

- suppose \( \text{sketch}(x_i) \leq \text{sketch}(q) < \text{sketch}(x_{i+1}) \)
- longest common prefix = lowest common ancestor between \( q \) & (either \( x_i \) or \( x_{i+1} \))
- nonsketch
- node \( y \) where \( q \) fell off paths to \( x_i \)'s
- if \( y \ell \) 1st bit of \( q \) is 1:
  - nearest \( x_i \) is in \( y0 \) subtree
  - nearest extreme in that subtree is \( e = y011 \ldots 1 \)
- else: \( e = y100 \ldots 0 \)

- predecessor & successor of \( q \) among \( x_i \)'s
- predecessor & successor of \( \text{sketch}(e) \) among \( \text{sketch}(x_i) \)'s

(in terms of \( \text{rank} \ i \sim \text{can translate to} \ x_i \)
Approximate sketch(\(x\)): on word RAM
- don't need sketch to pack \(b_i\) bits consecutively
- can spread out in predictable pattern of length \(O(w^{4/5})\)
  \[ \text{independent of } x \]

Idea: mask important bits: \(x' = x \text{ AND } \sum_{i=0}^{r-1} 2^{bi} \)
& multiply \(x'.m = (\sum_{i=0}^{r-1} x_{bi} 2^{bi}) (\sum_{j=0}^{r-1} 2^{mj}) \]
\[ = \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} x_{bi} 2^{bi+mj} \]

Claim: for any \(b_0, b_1, \ldots, b_{r-1}\), can choose \(m_0, m_1, \ldots, m_{r-1}\)
such that \(\oplus b_i + m_j \) are all distinct (no collision)
\(b_0 + m_0 < \cdots < b_{r-1} + m_{r-1}\) (preserve order)
\( (b_{r-1} + m_{r-1}) - (b_0 + m_0) = O(r^4) = O(w^{4/5}) \) (small)
\( \Rightarrow \text{approx-sketch}(x) = \left[ (x.m) \text{ AND } \sum_{i=0}^{r-1} 2^{bi+m}_i \right] >> (b_0 + m_0) \)

Proof: 1) choose \(m'_0, m'_1, \ldots, m'_{r-1} < r^3 \) such that
\( b_i + m'_j \) are all distinct \(\mod r^3 \) (strong @)
- pick \(m'_0, m'_1, \ldots, m'_{t-1}\) by induction
- \(m'_t\) must avoid \(b_j - b_k \text{ and } i, j, k \)
\( \Rightarrow \text{choice for } m'_t \text{ exists} \)
- to make nonnegative

2) let \(m_i = m'_i + (w - b_i + ir^3 \text{ rounded down to mult. of } r^3) \)
\( \equiv m'_i \text{ (mod } r^3) \)
\( \Rightarrow m_i + b_i \text{ in } r^3 \text{ interval after } (\lfloor \frac{w}{r^3} \rfloor + i) \cdot r^3 \)
\( \Rightarrow m_0 + b_0 < \cdots < m_{r-1} + b_{r-1} \)
\( \approx w \approx w + r^4 \Rightarrow \text{diff. } = O(r^4) \)
\[ \Box \]
Parallel comparison:
- sketch(node) = \( 1 \) sketch(x_0) \( \cdots \) 1 sketch(x_{k-1})
- sketch(q)^k = 0 sketch(q) \( \cdots \) 0 sketch(q)
- difference = \( \frac{1}{q} \) 00000 \( \cdots \) \( \frac{1}{q} \) 00001
- And with \( \frac{1}{q} \) 00000 \( \cdots \) \( \frac{1}{q} \) 00000
  \[
  \begin{align*}
  1 & \text{ if sketch(q) } \leq \text{ sketch(x_i)} \\
  0 & \text{ if sketch(q) } > \text{ sketch(x_i)}
  \end{align*}
  \]
  \( \Rightarrow \) these bits look like 00000111
  where sketch(q) fits \( \uparrow \)
  need index of most sig. 1 bit
- multiply with 0 00001 \( \cdots \) 0 00001
  \( \Rightarrow \) #’s \( \frac{1}{q} \)’s \( \text{desired} \)
  \( \text{desired} \) \text{ to right}
- AND with 11111 & shift right to get # 1’s
  = index of \( \emptyset \rightarrow 1 \) transition
  = k-rank in sketch world
- special case of:

Index of most significant 1 bit: 00010110 \( \Rightarrow \) 4
- \( AC^0 \) operation [Andersson, Miltersen, Thorup 1999]
- instruction on most modern CPUs
  (see Linux kernel: include/asm-*/*bitops.h; GCC: --builtin-clz; VC++: _BitScanReverse)
- needed during desketchifying (q XOR x_{i+1})
Word RAM solution: [Fredman & Willard 1993]
- Split word into $\sqrt{W}$ clusters of $\sqrt{W}$ bits each:
  
  \[ x = 0101 \ 0000 \ 1000 \ 1101 \]

- Similar to van Emde Boas, but no recursion
- Identify first nonempty cluster, then first 1 within

1. Identify nonempty clusters
   - AND $x$ with $F = 1000 \ 1000 \ 1000 \ 1000$
     \[ \Rightarrow 0000 \ 0000 \ 1000 \ 1000 \]
     = which clusters have first bit set
   - XOR with $x$ \[ \Rightarrow 0101 \ 0000 \ 0000 \ 0101 \]
     = remaining bits
   - Subtract $F$ - this:
     \[ 0*** \ 1000 \ 1000 \ 0*** \]
     borrow \(\Rightarrow\) nonempty
     \[ \Rightarrow\] no borrow \(\Rightarrow\) subtract \(\varnothing\)
   - AND with $F$ \(\Rightarrow 0000 \ 1000 \ 1000 \ 0000 \)
   - XOR with $F$ \(\Rightarrow 1000 \ 0000 \ 0000 \ 1000 \)
     nonempty \(\Rightarrow\) empty
   - OR with which clusters have first bit set
     \[ \Rightarrow y = 1000 \ 0000 \ 1000 \ 1000 \]
     = which clusters are nonempty
perfect sketch of $y$
- $b_i = \sqrt{w} - 1 + i \sqrt{w}$
- use $m_j = w - (\sqrt{w} - 1) - j \sqrt{w} + j$
$\Rightarrow b_i + m_j = w + (i-j)\sqrt{w} + j$ are unique for $0 \leq i, j < \sqrt{w}$
$\& b_i + m_i = w + i$
$\Rightarrow$ bits $w, w+1, \ldots, w+\sqrt{w}-1$ of $y \cdot m$
(shifted right $w$) form perfect sketch($y$)

find first 1 bit in sketch($y$)
= first nonempty cluster $c$
- use parallel comparison to find rank among:
  \[
  \begin{bmatrix}
  0001 \\
  0010 \\
  0100 \\
  1000 \\
  \end{bmatrix}
  \] $\sqrt{w}$ powers of $2$
- fits: $\sqrt{w} \cdot (\sqrt{w}+1) < 2w$ bits

find first 1 bit $d$ in identified cluster $c$
- shift right $c \cdot \sqrt{w}$ & AND with $1111$
to obtain cluster
- use parallel comparison as in 3

answer = $c \sqrt{w} + d$