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 – JCSS 1990]
- store n w-bit integers – here, statically
- $O(\log w n)$ time for predecessor/successor
- $O(n)$ space
- word RAM

$\Rightarrow$ predecessor $\leq \min \left\{ \log w n, \log w^3 \right\}$
\[
\leq \sqrt{\log n}
\]

- $\text{AC}^0$ RAM version [Andersson, Miltersen, Thorup – TCS]
  - ops. are constant-depth (unbounded fan) circuits
  - no multiplication

- dynamic version via exponential trees:
  $O(\log w n + \log \log n)$ deterministic updates
  [Andersson & Thorup – JACM 2007]

- dynamic version via hashing: [Raman – ESA 1996]
  $O(\log w n)$ expected updates

- \textbf{OPEN}:
  $O(\log w n)$ w.h.p. updates?
Idea: B-tree with branching factor $O(w^{1/5})$
\[ \Rightarrow \text{height} = \Theta(\log_w n) = \Theta(\log n / \log w) \]
- 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
\( \Psi \leq k-1 \) 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) \( \text{sketch}(x) = \text{extract bits } b_0, b_1, \ldots, b_{r-1} \) from \( x \)
i.e. \( r \)-bit vector whose \( i \)th bit = \( b_i \)th bit of word \( x \)
\( \Rightarrow \) \( \text{sketch}(x_0) < \text{sketch}(x_1) < \cdots < \text{sketch}(x_{k-1}) \)
& 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 \( \text{sketch}(q) \)
in parallel to \( \text{sketch}(x_0), \ldots, \text{sketch}(x_{k-1}) \)
- again \( AC^0 \) operation on \( O(1) \) words
  & we'll see a nice way with standard ops.
\( \Rightarrow \) find where \( \text{sketch}(q) \) fits among \( \text{sketch}(x_0) < \cdots < \text{sketch}(x_{k-1}) \)
- 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 \( x_{i+1} \))
- Node \( y \) where \( q \) fell off paths to \( x_i \)'s
- If \( l_y l+1 \text{st bit of } q \) is 1:
  - Nearest \( x_i \) is in \( y0 \) subtree
  - Nearest extreme in that subtree is \( e = y011\ldots1 \)
- Else: \( e = y100\ldots0 \)

- Predecessor & successor of \( q \) among \( x_i \)'s
- Predecessor & successor of \( \text{sketch}(e) \) among \( \text{sketch}(x_i) \)’s
  (in terms of rank, \( i \) ~ 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: \( x' = x \text{ AND } \sum_{i=0}^{r-1} 2^{b_i} \)
& multiply \( x', m = \left( \sum_{i=0}^{r-1} x_{bi} 2^{bi} \right) \left( \sum_{j=0}^{r-1} 2^{mj} \right) = \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} x_{bi} 2^{bi+mj} \)

Claim: for any \( b_0, b_1, \ldots, b_{r-1} \), can choose \( m_0, m_1, \ldots, m_{r-1} \)
such that
- \( b_i + m_j \) are all distinct
- \( b_0 + m_0 < \cdots < b_{r-1} + m_{r-1} \)
- \( (b_{r-1} + m_{r-1}) - (b_0 + m_0) = O(r^4) = O(w^{4/5}) \) (small)

\( \Rightarrow \) approx-sketch \( x = \left( (x, m) \text{ AND } \sum_{i=0}^{r-1} 2^{b_i+m_i} \right) \gg \left( b_0 + m_0 \right) \)

Proof: \( \Box \)

- \( \text{for different } i, j \text{ if } j \text{'s match, may have } b_i \equiv b_i \text{ mod } r^3 \text{ but still } b_i \neq b_i \text{' as needed} \)

- \( \text{pick } m_0, m_1, \ldots, m_{r-1} \text{ by induction} \)
- \( m_i \text{ must avoid } m_i + b_j - b_k \forall i, j, k \)

\( \Rightarrow \) choice for \( m_i \) exists

- to make nonnegative

\( \Rightarrow m_i + b_i \text{ in } r^3 \text{ interval after } (\lfloor \frac{w}{r^3} \rfloor + i) \cdot r^3 \)

\( \Rightarrow m_0 + b_0 < \cdots < m_{r-1} + b_{r-1} \)

\( \approx w \approx w + r^4 \Rightarrow \text{diff: } O(r^4) \)
Parallel comparison:
- sketch\((node)\) = \(\frac{1}{2}\) sketch\((x_0)\) \(\cdots\) \(\frac{1}{2}\) sketch\((x_{k-1})\)
- sketch\((q)^k\) = 0 sketch\((q)\) \(\cdots\) 0 sketch\((q)\)
- difference = \(\frac{1}{2}\) \(*\cdots\)* \(\frac{1}{2}\) \(*\cdots\)*
- And with
  \(\Rightarrow\)
  \[
  \begin{array}{c}
  1 \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \end{array}
  \begin{array}{c}
  00000 \cdots 00000 \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \end{array}
  \]
  \[
  \begin{array}{c}
  1 \\
  0 \\
  0 \\
  0 \\
  0 \\
  0 \\
  \end{array}
  \begin{array}{c}
  00000 \cdots 00000 \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \frac{1}{2} \\
  \end{array}
  \]

\(\Rightarrow\) these bits look like 0000\(\boxed{111}\)
where sketch\((q)\) fits

\(\Rightarrow\) need index of most sig. 1 bit

multiply with 0 00001 \(\cdots\) 0 00001
\[
\begin{array}{c}
\#1's \\
\#1's to right \\
\text{desired} \\
\text{last 1}
\end{array}
\]

\(\Rightarrow\) AND with 1111 & 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: 0001\(\boxed{0110}\) \(\Rightarrow\) 4

\(-\) AC\(\text{O}\) operation [Andersson, Miltersen, Thorup 1999]
\(-\) instruction on most modern CPUs
(see Linux kernel: include/asmp-x.h/bitops.h; GCC: -m 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:
  \[ x = 0101 \quad 0000 \quad 1000 \quad 1101 \]

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

1. Identify nonempty clusters
   - AND $x$ with $F = 1000 \quad 1000 \quad 1000 \quad 1000$
     \[ \Rightarrow \quad 0000 \quad 0000 \quad 1000 \quad 1000 \]
     = which clusters have first bit set
   - XOR with $x$ \[ \Rightarrow 0101 \quad 0000 \quad 0000 \quad 0101 \]
     = remaining bits
   - Subtract $F$ - this:
     \[ 0^{***} \quad 1000 \quad 1000 \quad 0^{***} \]
     \[ \Rightarrow \text{no borrow} \Rightarrow \text{subtract } \emptyset \]
   - AND with $F$ \[ \Rightarrow 0000 \quad 1000 \quad 1000 \quad 0000 \]
   - XOR with $F$ \[ \Rightarrow 1000 \quad 0000 \quad 0000 \quad 1000 \]
     = empty
   - OR with which clusters have first bit set
     \[ \Rightarrow y = 1000 \quad 0000 \quad 1000 \quad 1000 \]
     = which clusters are nonempty
2) 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 \( 0 \leq i, j < \sqrt{w} \)

& \( b_i + m_i = w + i \)
\[ \Rightarrow \text{bits } w, w+1, \ldots, w+\sqrt{w}-1 \text{ of } y \cdot m \]
(shifted right \( w \)) form perfect-sketch(\( y \))

3) find first 1 bit in sketch(\( y \))
- first nonempty cluster \( c \)
- use parallel comparison
  to find rank among:
\[ \{ 00001, 00100, 01100, 10000 \} \text{ powers of } 2 \] \[ \sqrt{w} \cdot (\sqrt{w}+1) < 2w \] bits

4) 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

5) answer = \( c \sqrt{w} + d \)