**Fusion trees:** [Fredman & Willard - JCSS 1993]
- store \( n \) \( w \)-bit integers - here, statically
- \( O(\log w n) \) time for predecessor/successor
- \( O(n) \) space
- dynamic version via exponential trees: \( O(\log w n + \lg \lg n) \) updates [Andersson & Thorup - JACM 2007]

**Consequence:** \( \min \{ \log w n, \lg w^3 \leq \sqrt{\lg n} \} \) \( \text{fusion} \) \( \text{van Emde Boas} \)
upper bound for predecessor problem

**Idea:** B-tree with branching factor \( \Theta(w^{1/5}) \)
- height = \( \Theta(\log w n) \)
  \( = \Theta(\lg n / \lg w) \)
- search must visit a node in \( O(1) \) time
- not enough time to read the node \( (w^{1/5} \ w\text{-bit words}) \) 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

\( \Rightarrow \leq k-1 \) 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) \( \text{sketch}(x) = \) 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 \( x_{i+1} \))

whichever’s longest/lowest

= Node \( y \) where \( q \) fell off paths to \( x_i \)’s
- If \( \|y\|+1 \)st bit of \( q \) is 1:
  - Nearest \( x_i \) is in \( y0 \) subtree
  - Nearest extreme in that subtree is \( e = y011...1 \)
- Else: \( e = y100...0 \)

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

(desketchified via \( \text{sketch}^{-1} \))
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 $x'.m = (\sum_{i=0}^{r-1} x_{b_i} 2^{b_i}) \cdot (\sum_{j=0}^{r-1} 2^{m_j})$
  $= \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} x_{b_i} 2^{b_i+m_j}$

Claim: for any $b_0, b_1, ..., b_{r-1}$ can choose $m_0, m_1, ..., m_{r-1}$ such that
  a) $b_i+m_j$ are all distinct (no collision)
  b) $b_0+m_0 < ... < b_{r-1}+m_{r-1}$ (preserve order)
  c) $(b_{r-1}+m_{r-1})-(b_0+m_0) = O(w^4) = O(w^{4/5})$ (small)

$\Rightarrow$ approx-sketch(x) = $[(x.m) \text{ AND } \sum_{i=0}^{r-1} 2^{b_i+m_i}] >> (b_0+m_0)$
  $\text{discard } i \neq j$

Proof: ① choose $m'_0, m'_1, ..., m'_{r-1} < r^3$ such that $b_i+m'_j$ are all distinct modulo $r^3$ (strong a)
  - pick $m'_0, m'_1, ..., m'_{r-1}$ by induction
  - $m'_k$ must avoid $m'_i + b_j - b_k \forall i, j, k$
  $r \times r \times \Rightarrow tr^2 < r^3$ choices

$\Rightarrow$ choice for $m'_k$ exists
  $\Rightarrow$ to make nonnegative

② let $m_i = m'_i + (\lfloor \frac{w}{r^3} \rfloor \times r^3 \text{ rounded down to mult of } r^3)$
  $\equiv m'_i \pmod{r^3}$

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

$m_0 + b_0 < \cdots < m_{r-1} + b_{r-1}$

$\equiv w \approx w + r^4 \Rightarrow \text{diff: } O(r^4)$ ⑥ ⑦
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 = \( \left(\frac{1}{\phi}\right) \) **** \( \cdots \) \( \left(\frac{1}{\phi}\right) \) ****
- And with
-\[ \rightarrow \left(\frac{1}{\phi}\right) \) 00000 \( \cdots \) \( \left(\frac{1}{\phi}\right) \) 00000

1 if sketch(q) \( \leq \) sketch(x\(_i\))
0 if sketch(q) \( > \) sketch(x\(_i\))

\[ \Rightarrow \text{these bits look like } 00000111 \]
where sketch(q\(_f\)) fits

new index of most sig. 1 bit

- multiply with
-\[ \rightarrow \left[\#1's\right] \] desired
\[ \left[\#1's\right] \text{ to right} \text{ last 1} \]

or if you prefer:

Index of most significant 1 bit: 00010110 \( \Rightarrow \) 4
- AC\(^o\) operation [Andersson, Miltersen, Thorup 1999]
- instruction on most modern CPUs
  (see Linux kernel: include/asm-*/*bitops.h)
- computable in O(1) using fusion tricks
  [Fredman & Willard 1993]