TODAY: van Emde Boas [Peter]
- series of improved data structures
- Insert, Successor
- Delete
- Space

Goal: maintain n elements among \{0, 1, \ldots, u-1\}
subject to Insert, Delete, Successor
in \(O(\log \log u)\) time/operation.

- if \(u = n^{O(1)}\) or \(n^{\log^{O(1)} n}\) then \(O(\log \log n)\) time/operation.!
- exponentially faster than balanced search trees
- cooler queries than hashing

Where might \(O(\log \log u)\) bound arise?
- binary search over \(\log u\) elements
- recurrences: \(T(\log u) = T(\log \frac{u}{2}) + O(1)\)
\(T(u) = T(5u^7) + O(1)\)

We'll develop van Emde Boas data structure
by a series of improvements.
1. **Bit vector**: \( V[x] = \) is \( x \) in the set?

   - **e.g.**
   
   \[
   \begin{array}{ccccccccccccc}
   0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
   0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
   \end{array}
   \]

   - **Insert/Delete**: \( O(1) \)
   - **Successor/Predecessor**: \( O(u) \)

2. **Split universe into clusters**: \( \sqrt{u} \) of size \( \sqrt{u} \)

   - **e.g.**
   
   \[
   \begin{array}{ccccccccccccc}
   0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
   0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
   \end{array}
   \]

   - if \( x = i \sqrt{u} + j \) then \( V[x] = V.cluster[i][j] \)

   \[0 \leq j < \sqrt{u}\]

   \[\Rightarrow \text{define} \begin{cases} 
   \text{low}(x) = x \mod \sqrt{u} = j \\
   \text{high}(x) = \lfloor x / \sqrt{u} \rfloor = i
   \end{cases}\]

   \( x: \begin{array}{ccc}
   \text{high}(x) & \text{low}(x) \\
   9 & 1 & 0 & 0 & 1
   \end{array}\)

   - index\((i, j)\) = \( i \sqrt{u} + j \)

   = high & low-order halves in binary

   - **Insert**: set \( V.cluster[\text{high}(x)][\text{low}(x)] \)

   \( \Rightarrow \text{mark cluster high}(x) \text{ nonempty} \)

   \( \Rightarrow O(1) \)

   - **Successor**: 
     - look within cluster high\((x)\)
     - else find next nonempty cluster i
     - find min \( j \) in that cluster
     - return index\((i, j)\)

   \( \Rightarrow O(u) \)
Recurse: 3 ops. in Successor are recursive Successors!
- \( V.cluster[i] = \text{size-}\sqrt{u} \text{ van Emde Boas} \), \( 0 \leq i < \sqrt{u} \)
- \( V.summary = \text{size-}\sqrt{u} \text{ van Emde Boas} \)
- \( V.summary[i] = \text{is } V.cluster[i] \text{ nonempty?} \)

\[
\text{Insert}(V, x):
\]
\[
\begin{align*}
\text{Insert}(V.cluster[\text{high}(x)], \text{low}(x)) \\
\text{Insert}(V.summary, \text{high}(x))
\end{align*}
\Rightarrow T(u) = 2T(\sqrt{u}) + O(1)
\]
\[
T'(\lg u) = 2T'(\frac{\lg u}{2}) + O(1)
\]
\[
= O(\lg u)
\]

\[
\text{Successor}(V, x):
\]
\[
i = \text{high}(x)
\]
\[
j = \text{Successor}(V.cluster[i], \text{low}(x))
\]
\[
\text{if } j = \infty:
\]
\[
i = \text{Successor}(V.summary, i)
\]
\[
j = \text{Successor}(V.cluster[i], -\infty)
\]
\[
\text{return } \text{index}(i, j)
\]
\[
\Rightarrow T(u) = 3T(\sqrt{u}) + O(1)
\]
\[
T'(\lg u) = 3T'(\frac{\lg u}{2}) + O(1)
\]
\[
= O((\lg u)\lg 3)
\]
\[
= O(\lg^{1.585} u)
\]

- need to reduce to one recursion!
4. Maintain \( \min \) & \( \max \) of every structure:

\(- O(1) \) overhead in Insert:
\[ \begin{align*}
& \text{if } x < V.\min: \ V.\min = x \\
& \text{if } x > V.\max: \ V.\max = x \\
\end{align*} \]

**Successor** \( (V, x) \):

\[ i = \text{high}(x) \]

\[ \text{if } \text{low}(x) < V.\text{cluster}[i].\max: \]
\[ j = \text{Successor}(V.\text{cluster}[i], \text{low}(x)) \]

\[ \text{else: } i = \text{Successor}(V.\text{summary}, \text{high}(x)) \]
\[ j = V.\text{cluster}[i].\min \]

\[ \text{return } \text{index}(i, j) \]

\[ \Rightarrow T(u) = T(\overline{u}) + O(1) \]
\[ = O(\log \log u) \]

5. Don't store \( \min \) recursively:

- Successor checks for \( \min \) specially:
\[ \text{if } x < V.\min: \text{ return } V.\min \]

**Insert** \((V, x)\):

\[ \text{empty case} \]
\[ \text{if } V.\min = \text{None}: \ V.\min = V.\max = x; \text{ return} \]
\[ \text{if } x < V.\min: \text{ swap } x \leftrightarrow V.\min \]
\[ \text{if } x > V.\max: \ V.\max = x \]
\[ \text{if } V.\text{cluster}[\text{high}(x)].\min = \text{None}: \]
\[ \text{Insert} (V.\text{summary}, \text{high}(x)) \]
\[ \text{Insert} (V.\text{cluster}[\text{high}(x)], \text{low}(x)) \]

\[ \ast \text{ if both calls, then second costs } O(1) \text{ (case)} \]

\[ \Rightarrow T(u) = O(\log \log u) \]
Delete \((V, x)\):

if \(x = V.\text{min}\):
    \(i = V.\text{summary}.\text{min}\)
    if \(i = \text{None}\):
        \(V.\text{min} = \text{None}\)  \(\{\text{empty now}\}\)
        return
    \(x = V.\text{min} = \text{index}(i, V.\text{cluster}[i].\text{min})\)  \(\{\text{unstore new min}\}\)
    Delete \((V.\text{cluster}[\text{high}(x)].\text{low}(x))\)
    if \(V.\text{cluster}[\text{high}(x)].\text{min} = \text{None}\):
        Delete \((V.\text{summary}.\text{high}(x))\)  \(\{\text{empty now}\}\)
    \(\{\text{second call}\}\)

\(\{\text{update max}\}\)

if \(x = V.\text{max}\):
    if \(V.\text{summary}.\text{max} = \text{None}\):
        \(V.\text{max} = V.\text{min}\)
    else:
        \(i = V.\text{summary}.\text{max}\)
        \(V.\text{max} = \text{index}(i, V.\text{cluster}[i].\text{max})\)
        \(\{\text{just min now}\}\)

\(*\) if make second call, then first call was cheap (just deleted a min)
\(\Rightarrow T(u) = O(\log \log u)\)

\underline{Lower bound:} [Pătraşcu & Thorup 2007]
\(\Omega(\log \log u)\) for \(u = n^{o(1)}\)
\& space = \(O(n \text{ polylog } n)\)
Space: can improve from current $\Theta(n)$ to $O(n)$
- only create nonempty clusters
  - if $V\.\text{min}$ becomes None, deallocate $V$
- $V\.\text{cluster} = \text{hash table of nonempty clusters}$
  $\Rightarrow$ space = $O(\# \text{nonempty “child clusters”})$
  (recall from 6.006; and see Lecture 10)
- charge each table entry $i$ to $V\.\text{cluster}[i].\text{min}$
  $\Rightarrow$ each element charged $\leq$ once
  (each cluster has only one “parent”)
- charge remaining $O(1)$ space of $V$
  (min, max, $O(1)$ pointers) to $V\.\text{min}$
  $\Rightarrow$ each element charged $\leq$ once more
  (actually twice: each elt. split into two min slots)
$\Rightarrow O(n)$ space (but randomized)

CHARGING AMORTIZATION ~
SEE NEXT LECTURE (5)