Fixed-universe successor problem

**Goal:** Maintain a dynamic subset $S$ of size $n$ of the universe $U = \{0, 1, \ldots, u - 1\}$ of size $u$ subject to these operations:

- **INSERT**$(x \in U \setminus S)$: Add $x$ to $S$.
- **DELETE**$(x \in S)$: Remove $x$ from $S$.
- **SUCCESSOR**$(x \in U)$: Find the next element in $S$ larger than any element $x$ of the universe $U$.
- **PREDECESSOR**$(x \in U)$: Find the previous element in $S$ smaller than $x$. 
Solutions to fixed-universe successor problem

**Goal:** Maintain a dynamic subset $S$ of size $n$ of the universe $U = \{0, 1, \ldots, u - 1\}$ of size $u$ subject to **INSERT**, **DELETE**, **SUCCESSOR**, **PREDECESSOR**.

- Balanced search trees can implement operations in $O(\lg n)$ time, without fixed-universe assumption.
- In 1975, Peter van Emde Boas solved this problem in $O(\lg \lg u)$ time per operation.
  - If $u$ is only polynomial in $n$, that is, $u = O(n^c)$, then $O(\lg \lg n)$ time per operation--exponential speedup!
$O(lg \ lg \ u)$?! 

Where could a bound of $O(lg \ lg \ u)$ arise?

- Binary search over $O(lg \ u)$ things

\[
T(u) = T(\sqrt{u}) + O(1)
\]
\[
T'(lg \ u) = T'((lg \ u)/2) + O(1)
\]
\[
= O(lg \ lg \ u)
\]
(1) Starting point: Bit vector

**Bit vector** $v$ stores, for each $x \in U$, 
\[ v_x = \begin{cases} 
1 & \text{if } x \in S \\
0 & \text{if } x \notin S 
\end{cases} \]

**Example**: $u = 16$; $n = 4$; $S = \{1, 9, 10, 15\}$.

Insert/Delete run in $O(1)$ time.
Successor/Predecessor run in $O(u)$ worst-case time.
### (2) Split universe into widgets

Carve universe of size $u$ into $\sqrt{u}$ widgets $W_0, W_1, \ldots, W_{\sqrt{u} - 1}$ each of size $\sqrt{u}$.

**Example:** $u = 16$, $\sqrt{u} = 4$.

<table>
<thead>
<tr>
<th></th>
<th>$W_0$</th>
<th>$W_1$</th>
<th>$W_2$</th>
<th>$W_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>2</td>
<td>4 5 6 7</td>
<td>4 5 6 7</td>
<td>4 5 6 7</td>
<td>4 5 6 7</td>
</tr>
<tr>
<td>3</td>
<td>8 9 10 11</td>
<td>8 9 10 11</td>
<td>8 9 10 11</td>
<td>8 9 10 11</td>
</tr>
<tr>
<td>4</td>
<td>12 13 14 15</td>
<td>12 13 14 15</td>
<td>12 13 14 15</td>
<td>12 13 14 15</td>
</tr>
</tbody>
</table>
(2) Split universe into widgets

Carve universe of size $u$ into $\sqrt{u}$ widgets $W_0, W_1, \ldots, W_{\sqrt{u} - 1}$ each of size $\sqrt{u}$.

$W_0$ represents $0, 1, \ldots, \sqrt{u} - 1 \in U$;
$W_1$ represents $\sqrt{u}, \sqrt{u} + 1, \ldots, 2\sqrt{u} - 1 \in U$;
$\vdots$
$W_i$ represents $i\sqrt{u}, i\sqrt{u} + 1, \ldots, (i + 1)\sqrt{u} - 1 \in U$;
$\vdots$
$W_{\sqrt{u} - 1}$ represents $u - \sqrt{u}, u - \sqrt{u} + 1, \ldots, u - 1 \in U$. 
(2) Split universe into widgets

Define \( \text{high}(x) \geq 0 \) and \( \text{low}(x) \geq 0 \) so that \( x = \text{high}(x) \sqrt{u} + \text{low}(x) \).
That is, if we write \( x \in U \) in binary, \( \text{high}(x) \) is the high-order half of the bits, and \( \text{low}(x) \) is the low-order half of the bits.
For \( x \in U \), \( \text{high}(x) \) is index of widget containing \( x \) and \( \text{low}(x) \) is the index of \( x \) within that widget.
(2) Split universe into widgets

\text{INSERT}(x)

insert $x$ into widget $W_{\text{high}(x)}$ at position $\text{low}(x)$.
mark $W_{\text{high}(x)}$ as nonempty.

Running time $T(n) = O(1)$.
(2) Split universe into widgets

\textbf{SUCCESSOR}(x)

- look for successor of \(x\) within widget \(W_{\text{high}(x)}\) starting after position \(\text{low}(x)\).

\begin{align*}
\text{if successor found} & \quad \text{then return it} \\
\text{else find smallest } i > \text{high}(x) & \quad \text{for which } W_i \text{ is nonempty.} \\
& \quad \text{return smallest element in } W_i
\end{align*}

\text{Running time } T(u) = O(\sqrt{u}).
Revelation

\text{Successor}(x) \\
\text{look for successor of } x \text{ within widget } W_{\text{high}(x)} \text{ starting after position } low(x). \\
\text{if} \text{ successor found} \\
\text{then return it} \\
\text{else} \text{ find smallest } i > high(x) \\
\text{for which } W_i \text{ is nonempty.} \\
\text{return smallest element in } W_i
(3) Recursion

Represent universe by \textit{widget} of size $u$. Recursively split each widget $W$ of size $|W|$ into $\sqrt{|W|}$ \textit{subwidgets} $\text{sub}[W][0]$, $\text{sub}[W][1]$, $\ldots$, $\text{sub}[W][\sqrt{|W|}-1]$ each of size $\sqrt{|W|}$. Store a \textit{summary widget} $\text{summary}[W]$ of size $\sqrt{|W|}$ representing which subwidgets are nonempty.
(3) Recursion

Define $high(x) \geq 0$ and $low(x) \geq 0$
so that $x = high(x)\sqrt{|W|} + low(x)$.

**INSERT**($x$, $W$)

if $sub[W][high(x)]$ is empty
then **INSERT**($high(x)$, $summary[W]$)
**INSERT**($low(x)$, $sub[W][high(x)]$)

Running time $T(u) = 2 \cdot T(\sqrt{u}) + O(1)$
$T'(\log u) = 2 \cdot T'((\log u) / 2) + O(1)$
$= O(\log u)$.
(3) Recursion

**SUCCESSOR**\((x, W)\)

\[
j \leftarrow \text{SUCCESSOR}(\text{low}(x), \text{sub}[W][\text{high}(x)])
\]

\[
\text{if } j < \infty \text{ then return } \text{high}(x) \sqrt{|W|} + j
\]

\[
\text{else } i \leftarrow \text{SUCCESSOR}(\text{high}(x), \text{summary}[W])
\]

\[
j \leftarrow \text{SUCCESSOR}(\infty, \text{sub}[W][i])
\]

\[
\text{return } i \sqrt{|W|} + j
\]

Running time \(T(u) = 3 T(\sqrt{u}) + O(1)\)

\[
T'(\lg u) = 3 T'((\lg u) / 2) + O(1)
\]

\[= O((\lg u)^{\lg 3}).\]
Improvements

Need to reduce INSERT and SUCCESSOR down to 1 recursive call each.

- 1 call: \( T(u) = 1 \ T(\sqrt{u}) + O(1) \)
  \[ = O(\lg \lg n) \]
- 2 calls: \( T(u) = 2 \ T(\sqrt{u}) + O(1) \)
  \[ = O(\lg n) \]
- 3 calls: \( T(u) = 3 \ T(\sqrt{u}) + O(1) \)
  \[ = O((\lg u)^{\lg 3}) \]

We’re closer to this goal than it may seem!
Recursive calls in successor

If \( x \) has a successor within \( \text{sub}[W][\text{high}(x)] \), then there is only 1 recursive call to SUCCESSOR. Otherwise, there are 3 recursive calls:

- \( \text{SUCCESSOR}(\text{low}(x), \text{sub}[W][\text{high}(x)]) \)
  discovers that \( \text{sub}[W][\text{high}(x)] \) hasn’t successor.
- \( \text{SUCCESSOR}(\text{high}(x), \text{summary}[W]) \)
  finds next nonempty subwidget \( \text{sub}[W][i] \).
- \( \text{SUCCESSOR}(\neg \infty, \text{sub}[W][i]) \)
  finds smallest element in subwidget \( \text{sub}[W][i] \).
Reducing recursive calls in successor

If $x$ has no successor within $\text{sub}[W][\text{high}(x)]$, there are 3 recursive calls:

- **SUCCESSOR**(low($x$), sub[$W$][high($x$))]) discovers that $\text{sub}[W][\text{high}(x)]$ hasn’t successor.
  - Could be determined using the maximum value in the subwidget $\text{sub}[W][\text{high}(x)]$.

- **SUCCESSOR**(high($x$), summary[$W$]) finds next nonempty subwidget $\text{sub}[W][i]$.

- **SUCCESSOR**($-\infty$, $\text{sub}[W][i]$) finds minimum element in subwidget $\text{sub}[W][i]$.
(4) Improved successor

\textbf{INSERT}(x, \ W)

\textbf{if} sub[\ W][\textit{high}(x)] \textbf{is empty}
\textbf{then} \textbf{INSERT} (\textit{high}(x), \summary[\ W])
\textbf{INSERT} (\textit{low}(x), \textit{sub}[\ W][\textit{high}(x)])
\textbf{if} x < \textit{min}[\ W] \textbf{then} \textit{min}[\ W] \leftarrow x
\textbf{if} x > \textit{max}[\ W] \textbf{then} \textit{max}[\ W] \leftarrow x

\textit{new (augmentation)}

\textbf{Running time} \quad T(u) = 2 \ T(\sqrt{u}) + O(1)
\quad \quad T'(\lg u) = 2 \ T'((\lg u) / 2) + O(1)
\quad \quad = O(\lg u) .
(4) Improved successor

\textbf{SUCCESSOR}(x, W)

\begin{align*}
\text{if } \text{low}(x) &< \max[\text{sub}[W][\text{high}(x)]] \\
\text{then } j &\leftarrow \text{SUCCESSOR(\text{low}(x), sub[W][\text{high}(x)]) } \} \quad T(\sqrt{u}) \\
\text{return } \text{high}(x)\sqrt{|W|} + j \\
\text{else } i &\leftarrow \text{SUCCESSOR(\text{high}(x), summary[W]) } \} \quad T(\sqrt{u}) \\
j &\leftarrow \min[\text{sub}[W][i]] \\
\text{return } i\sqrt{|W|} + j
\end{align*}

Running time \( T(u) = 1 \ T(\sqrt{u}) + O(1) \)
\[= O(\lg \lg u) \, . \]
Recursive calls in insert

If \( sub[W][\text{high}(x)] \) is already in \( summary[W] \), then there is only 1 recursive call to \textsc{Insert}. Otherwise, there are 2 recursive calls:

- \textsc{Insert}(\text{high}(x), summary[W])
- \textsc{Insert}(\text{low}(x), sub[W][\text{high}(x)])

\textbf{Idea:} We know that \( sub[W][\text{high}(x)] \) is empty. Avoid second recursive call by specially storing a widget containing just 1 element. Specifically, do not store \textit{min} recursively.
(5) Improved insert

\[ \text{Insert}(x, W) \]
\[
\text{if } x < \text{min}[W] \text{ then exchange } x \leftrightarrow \text{min}[W] \\
\text{if } \text{sub}[W][\text{high}(x)] \text{ is nonempty, that is,} \\
\hspace{1cm} \text{min}[\text{sub}[W][\text{high}(x)] \neq \text{NIL} \\
\text{then } \text{Insert}(\text{low}(x), \text{sub}[W][\text{high}(x)]) \\
\text{else } \text{min}[\text{sub}[W][\text{high}(x)] \leftarrow \text{low}(x) \\
\hspace{1cm} \text{Insert}(\text{high}(x), \text{summary}[W]) \\
\text{if } x > \text{max}[W] \text{ then } \text{max}[W] \leftarrow x
\]

Running time \( T(u) = 1 \ T(\sqrt{u}) + O(1) \)
\[
= O(\lg \lg u) .
\]
(5) Improved insert

\texttt{SUCCESSOR}(x, W)
\begin{align*}
\text{if } x < \text{min}[W] \text{ then return } \text{min}[W] \quad \text{ new} \\
\text{if } \text{low}(x) < \text{max}[\text{sub}[W][\text{high}(x)]] \\
\text{then } j \leftarrow \text{SUCCESSOR}(\text{low}(x), \text{sub}[W][\text{high}(x)]) \\
\text{return } \text{high}(x) \sqrt{|W|} + j \\
\text{else } i \leftarrow \text{SUCCESSOR}(\text{high}(x), \text{summary}[W]) \\
\quad j \leftarrow \text{min}[\text{sub}[W][i]] \\
\quad \text{return } i \sqrt{|W|} + j
\end{align*}

Running time $T(u) = 1 \ T(\sqrt{u}) + O(1) = O(\lg \lg u)$.
Deletion

\textbf{DELETE}(x, W)

\begin{enumerate}
\item if \(\text{min}[W] = \text{NIL}\) or \(x < \text{min}[W]\) then return
\item if \(x = \text{min}[W]\)
\item \(i \leftarrow \text{min}[\text{summary}[W]]\)
\item \(x \leftarrow i \sqrt{|W|} + \text{min}[\text{sub}[W][i]]\)
\item \(\text{min}[W] \leftarrow x\)
\item \text{DELETE}(\text{low}(x), \text{sub}[W][\text{high}(x)])\)
\item if \(\text{sub}[W][\text{high}(x)]\) is now empty, that is, \(\text{min}[\text{sub}[W][\text{high}(x)] = \text{NIL}\)
\item \text{DELETE}(\text{high}(x), \text{summary}[W])\)
\end{enumerate}

\textit{(in this case, the first recursive call was cheap)}