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 $U = \{0, 1, ..., u - 1\}$ 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!
Where could a bound of $O(lg \ lg \ u)$ arise?

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

Let $T(u) = T(\sqrt{u}) + O(1)$

$T'(lg \ u) = T'((lg \ u)/2) + O(1)$

$= O(lg \ lg \ u)$
(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**: \( u = 16; \ n = 4; \ S = \{1, 9, 10, 15\} \).

\[
\begin{array}{cccccccccccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{array}
\]

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>$W_0$</th>
<th>$W_1$</th>
<th>$W_2$</th>
<th>$W_3$</th>
</tr>
</thead>
<tbody>
<tr>
<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>0 1 2 3</td>
<td>4 5 6 7</td>
<td>8 9 10 11</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

\textbf{Insert}(x)

\begin{itemize}
  \item insert \( x \) into widget \( W_{\text{high}(x)} \) at position \( \text{low}(x) \).
  \item mark \( W_{\text{high}(x)} \) as nonempty.
\end{itemize}

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

**SUCCESSOR**(x)

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

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

**SUCCESSOR**(x)

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

if successor found

then return it

else find smallest $i > \text{high}(x)$ for which $W_i$ is nonempty.

return smallest element in $W_i$
(3) Recursion

Represent universe by *widget* of size \( u \).
Recursively split each widget \( W \) of size \(|W|\) into \( \sqrt{|W|} \) *subwidgets* \( \text{sub}[W][0], \text{sub}[W][1], \ldots, \text{sub}[W][\sqrt{|W|} - 1] \) each of size \( \sqrt{|W|} \).
Store a *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 $T(u) = 2 \ T(\sqrt{u}) + O(1)$

$T'(\lg u) = 2 \ T'((\lg u) / 2) + O(1)$

$= O(\lg u)$.
(3) Recursion

\textbf{SUCCESSOR}(x, W)

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

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 \texttt{INSERT} and \texttt{SUCCESSOR} down to 1 recursive call each.

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

\textit{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:

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

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

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

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

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

\textbf{INSERT}(x, W)

\begin{enumerate}
\item \textbf{if} \textit{sub}[$W$][\textit{high}(x)] is empty \textbf{then} \textbf{INSERT}(\textit{high}(x), summary[$W$])
\item \textbf{INSERT}(\textit{low}(x), \textit{sub}[$W$][\textit{high}(x)])
\end{enumerate}

\textbf{if} \textit{x} < \textit{min}[$W$] \textbf{then} \textit{min}[$W$] \leftarrow \textit{x}
\textbf{if} \textit{x} > \textit{max}[$W$] \textbf{then} \textit{max}[$W$] \leftarrow \textit{x}

\textbf{Running time} \quad T(u) = 2 \ T(\sqrt{u}) + O(1)

\text{ } \quad T'(\lg u) = 2 \ T'((\lg u) / 2) + O(1)

\text{ } \quad = O(\lg u) .
(4) Improved successor

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

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

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

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

\textbf{Idea:} We know that $\text{sub}[W][\text{high}(x)]$ 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

\textbf{Insert}(x, \ W)

\begin{itemize}
  \item if $x < \text{min}[W]$ then exchange $x \leftrightarrow \text{min}[W]$
  \item if $\text{sub}[W][\text{high}(x)]$ is nonempty, that is, $\text{min}[\text{sub}[W][\text{high}(x)] \neq \text{NIL}$
    \begin{itemize}
      \item then \textbf{Insert}(\text{low}(x), \text{sub}[W][\text{high}(x)])
    \end{itemize}
  \item else $\text{min}[\text{sub}[W][\text{high}(x)]] \leftarrow \text{low}(x)$
    \begin{itemize}
      \item \textbf{Insert}(\text{high}(x), \text{summary}[W])
    \end{itemize}
  \item if $x > \text{max}[W]$ then $\text{max}[W] \leftarrow x$
\end{itemize}

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

\textbf{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 \quad \text{new} \\
\text{else } i & \leftarrow \text{SUCCESSOR}(\text{high}(x), \text{summary}[W]) \\
& j \leftarrow \text{min}[\text{sub}[W][i]] \\
& \text{return } i \sqrt{|W|} + j
\end{align*}

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

\texttt{DELETE}(x, W)

\begin{itemize}
  \item if \( \text{min}[W] = \text{NIL} \) or \( x < \text{min}[W] \) then return
  \item if \( x = \text{min}[W] \)
    \begin{itemize}
      \item then \( i \leftarrow \text{min}[\text{summary}[W]] \)
      \item \( x \leftarrow i \sqrt{|W|} + \text{min}[\text{sub}[W][i]] \)
      \item \( \text{min}[W] \leftarrow x \)
    \end{itemize}
  \item \texttt{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 then \texttt{DELETE}(\text{high}(x), \text{summary}[W])
\end{itemize}

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