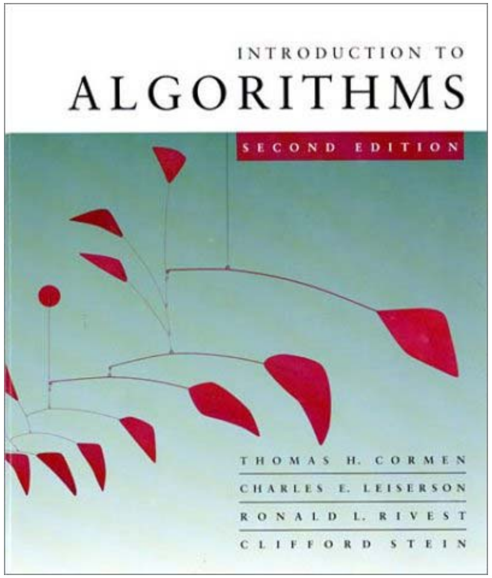
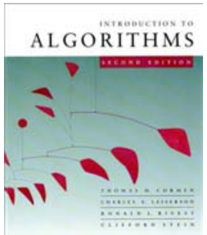
Introduction to Algorithms

6.046J/18.401J/SMA5503



Lecture 14

Prof. Erik Demaine



Fixed-universe successor problem

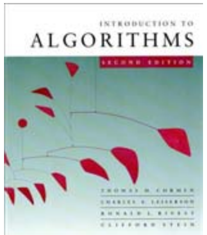
Goal: Maintain a dynamic subset S of size n of the universe $U = \{0, 1, \ldots, u - 1\}$ of size u subject to these operations:

- **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!



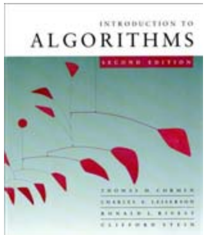
$O(\lg \lg u)$?!

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

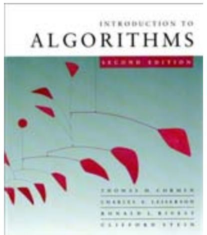
Bit vector v stores, for each $x \in U$,

$$v_x = \begin{cases} 1 \text{ if } x \in S \\ 0 \text{ if } x \notin S \end{cases}$$

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

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

Insert/Delete run in $O(1)$ time.
Successor/Predecessor run in $O(u)$ worst-case time.



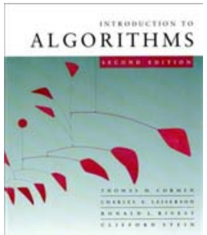
(2) Split universe into widgets

Carve universe of size u into $\sqrt{u}$ widgets

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

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

W_0				W_1				W_2				W_3			
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



(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 $0, 1, \ldots, \sqrt{u}-1 \in U$;

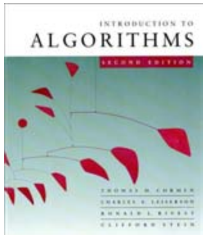
W_1 represents $\sqrt{u}, \sqrt{u}+1, \ldots, 2\sqrt{u}-1 \in U$;

$\vdots$

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 $high(x) \ge 0$ and $low(x) \ge 0$ so that $x = high(x)\sqrt{u} + low(x)$.

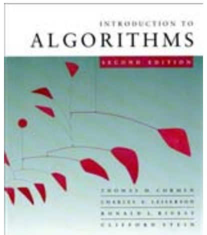
That is, if we write $x \in U$ in binary, $high(x)$ is the high-order half of the bits, and $low(x)$ is the low-order half of the bits.

$$x = 9$$

$$\underbrace{\boxed{1}\,\boxed{0}}_{\substack{high(x)\\ = 2}}\ \underbrace{\boxed{0}\,\boxed{1}}_{\substack{low(x)\\ = 1}}$$

For $x \in U$, $high(x)$ is index of widget containing x and $low(x)$ is the index of x within that widget.

W_0				W_1				W_2				W_3			
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



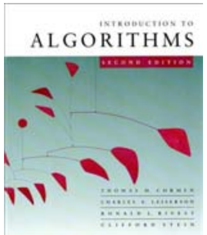
(2) Split universe into widgets

INSERT(x)

insert x into widget $W_{high(x)}$ at position $low(x)$.

mark $W_{high(x)}$ as nonempty.

Running time $T(n) = O(1)$.



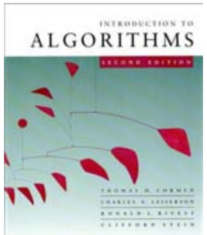
(2) Split universe into widgets

SUCCESSOR(x)

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

if successor found
 then return it
 else find smallest $i > high(x)$ for which W_i is nonempty. — $O(\sqrt{u})$
 return smallest element in W_i — $O(\sqrt{u})$

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

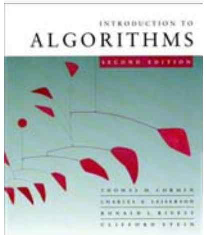


Revelation

SUCCESSOR(x)

    look for successor of x within widget $W_{high(x)}$
        starting after position $low(x)$. — *recursive successor*

    if successor found
     then return it
     else find smallest $i > high(x)$
                for which W_i is nonempty. — *recursive successor*
          return smallest element in W_i — *recursive successor*

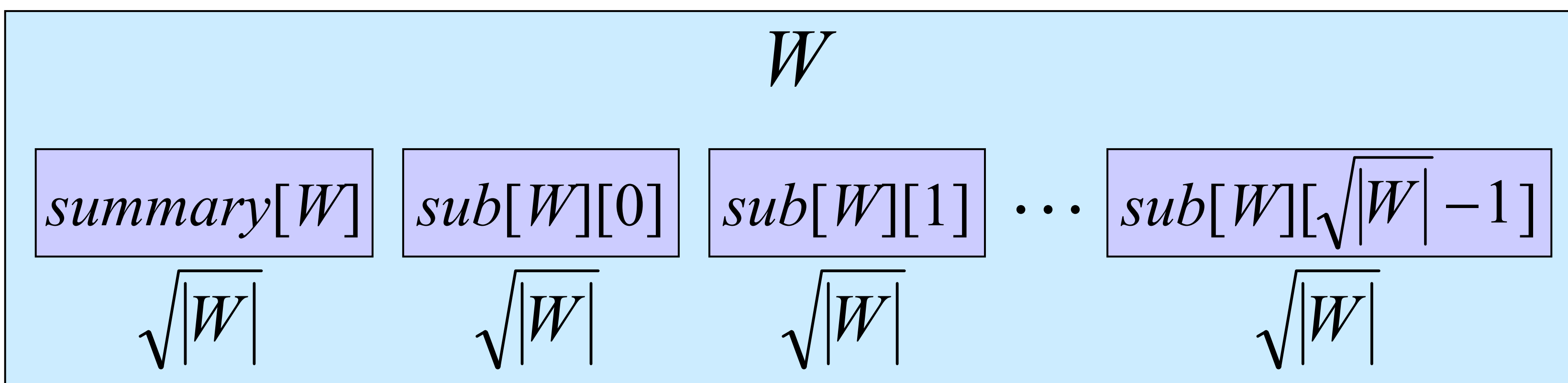


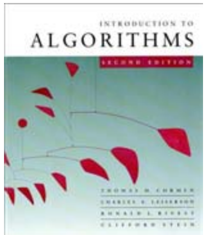
(3) Recursion

Represent universe by ***widget*** of size u.

Recursively split each widget W of size $|W|$ into $\sqrt{|W|}$ ***subwidgets*** $sub[W][0], sub[W][1], \ldots, sub[W][\sqrt{|W|}-1]$ each of size $\sqrt{|W|}$.

Store a ***summary widget*** $summary[W]$ 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

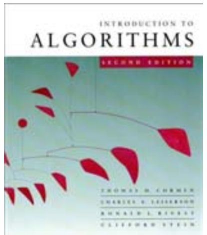
$\text{SUCCESSOR}(x, W)$

$$
\begin{array}{ll}
\quad j \leftarrow \text{SUCCESSOR}(low(x), sub[W][high(x)]) & \} \; T(\sqrt{u}) \\
\quad \textbf{if } j < \infty & \\
\quad\quad \textbf{then return } high(x)\sqrt{|W|} + j & \\
\quad\quad \textbf{else } i \leftarrow \text{SUCCESSOR}(high(x), summary[W]) & \} \; T(\sqrt{u}) \\
\quad\quad\quad\quad j \leftarrow \text{SUCCESSOR}(-\infty, sub[W][i]) & \} \; T(\sqrt{u}) \\
\quad\quad\quad\quad \textbf{return } i\sqrt{|W|} + j &
\end{array}
$$

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.

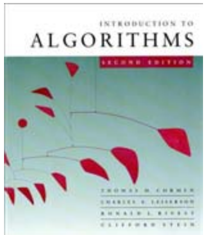
- 1 call: $T(u) = 1\,T(\sqrt{u}) + O(1)$
  $= O(\lg \lg n)$
- 2 calls: $T(u) = 2\,T(\sqrt{u}) + O(1)$
  $= O(\lg n)$
- 3 calls: $T(u) = 3\,T(\sqrt{u}) + O(1)$
  $= O((\lg u)^{\lg 3})$

We're closer to this goal than it may seem!

Recursive calls in successor

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

- SUCCESSOR$(low(x), sub[W][high(x)])$ discovers that $sub[W][high(x)]$ hasn't successor.
- SUCCESSOR$(high(x), summary[W])$ finds next nonempty subwidget $sub[W][i]$.
- SUCCESSOR$(-\infty, sub[W][i])$ finds smallest element in subwidget $sub[W][i]$.



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

INSERT(x, W)

if $sub[W][high(x)]$ is empty

then INSERT$(high(x), summary[W])$

INSERT$(low(x), sub[W][high(x)])$

if $x < min[W]$ **then** $min[W] \leftarrow x$

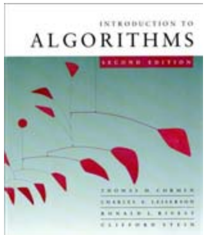
if $x > max[W]$ **then** $max[W] \leftarrow x$

new (augmentation)

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

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

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



(4) Improved successor

SUCCESSOR(x, W)

if $low(x) < max[sub[W][high(x)]]$

then $j \leftarrow$ SUCCESSOR$(low(x), sub[W][high(x)])$ $\}$ $T(\sqrt{u})$

return $high(x)\sqrt{|W|} + j$

else $i \leftarrow$ SUCCESSOR$(high(x), summary[W])$ $\}$ $T(\sqrt{u})$

$j \leftarrow min[sub[W][i]]$

return $i\sqrt{|W|} + j$

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

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

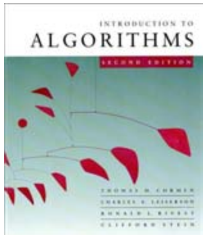
Recursive calls in insert

If $sub[W][high(x)]$ is already in $summary[W]$, then there is only 1 recursive call to INSERT.

Otherwise, there are 2 recursive calls:

- INSERT$(high(x), summary[W])$
- INSERT$(low(x), sub[W][high(x)])$

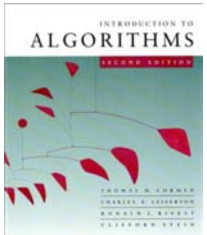
Idea: We know that $sub[W][high(x)])$ is empty.
Avoid second recursive call by specially storing a widget containing just 1 element.
Specifically, do not store *min* recursively.



(5) Improved insert

INSERT(x, W)
  if $x < min[W]$ **then** exchange $x \leftrightarrow min[W]$
  if $sub[W][high(x)]$ is nonempty, that is,
    $min[sub[W][high(x)]] \neq$ NIL
   then INSERT$(low(x), sub[W][high(x)])$
   else $min[sub[W][high(x)]] \leftarrow low(x)$
      INSERT$(high(x), summary[W])$
  if $x > max[W]$ **then** $max[W] \leftarrow x$

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



(5) Improved insert

SUCCESSOR(x, W)

  if $x < min[W]$ **then return** $min[W]$ } new

  if $low(x) < max[sub[W][high(x)]]$ } $T(\sqrt{u})$

   then $j \leftarrow$ SUCCESSOR$(low(x), sub[W][high(x)])$

      return $high(x)\sqrt{|W|} + j$ } $T(\sqrt{u})$

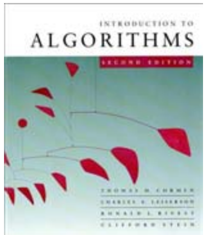
   else $i \leftarrow$ SUCCESSOR$(high(x), summary[W])$

     $j \leftarrow min[sub[W][i]]$

     return $i\sqrt{|W|} + j$

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

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



Deletion

DELETE(x, W)

  if $min[W] = \text{NIL}$ or $x < min[W]$ **then return**

  if $x = min[W]$

   then $i \leftarrow min[summary[\text{W}]]$

     $x \leftarrow \text{i}\sqrt{|W|} + min[sub[W][i]]$

     $min[W] \leftarrow x$

  DELETE($low(x)$, $sub[W][high(x)]$)

  if $sub[W][high(x)]$ is now empty, that is,

   $min[sub[W][high(x)]] = \text{NIL}$

  then DELETE($high(x)$, $summary[W]$)

     (in this case, the first recursive call was cheap)