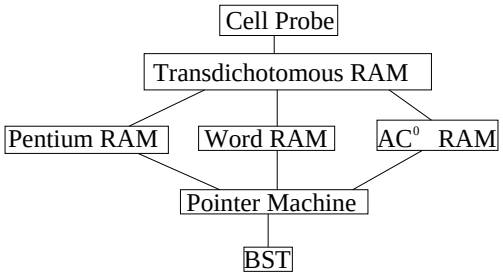
Lecture 12 Integer Data Structures

- hashing: O(1) for membership. Beat $O(\lg n)$ for *successor* and *predecessor* queries.
- elements (inputs, outputs, memory cells) are w-bit integers from universe $\mathcal{U} = \{0, 1, \ldots, u-1\}$, $u = 2^w$.

Models of Computation

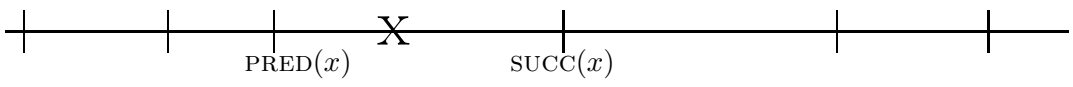


- *cell probe model:* Only pay for memory access. Non-realistic, but good for proving lower bounds.
- *transdichotomous RAM:* integers can be used as pointers ($w \geq \lg n$). "computer" changes with the problem size...
  $\Rightarrow$ idea: necessary just to address the input.
- *word RAM:* transdichotomous + only "C-style" operations (`+ - * / % & | ^ ~ << >>`).
- *AC^0 RAM:* transdichotomous + only "AC^0" operations (constant depth, poly sized circuits with unbounded fan-in)
  - excludes multiplication.
- *Pointer-machine:* directed graph, constant branching factor.

Predecessor/Successor problem

Maintain a set S of n w-bit integers from a universe $\mathcal{U} = \{0, 1, \ldots, u-1\}$ of size $u = 2^w$ (word RAM) under:

- INSERT(x), $\quad x \in \mathcal{U}$
- DELETE(x), $\quad x \in S$
- SUCCESSOR(x), $\quad x \in \mathcal{U}$
- PREDECESSOR(x), $\quad x \in \mathcal{U}$



Classic Results

	data structure	time/op	space	model
1962	BST's	$O(\lg n)$	$O(n)$	BST
1975	van Emde Boas [1]	$O(\lg \lg u)$	$O(u)$	word / AC^0 RAM
1983	y-fast tries [2]	$O(\lg \lg u)$ w.h.p.	$O(n)$	word RAM
1993	fusion trees [3]	$O\left(\frac{\lg n}{\lg \lg u}\right)$	$O(n)$	word / AC^0 RAM [4]

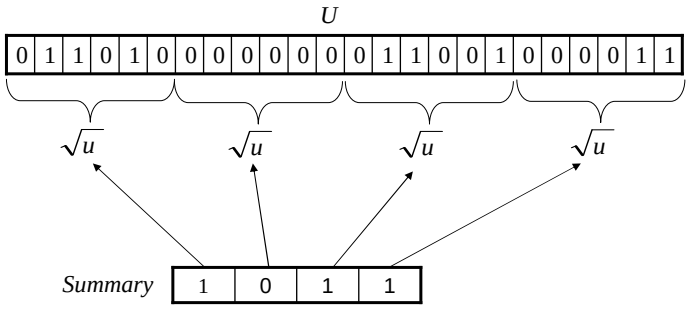
Combination $O(\sqrt{\lg n})$:
If $\lg \lg u \leq \sqrt{\lg n}$ then use van Emde Boas or y-fast tries. Otherwise, use fusion trees: $O\left(\frac{\lg n}{\lg \lg u}\right) \leq O(\sqrt{\lg n})$.

van Emde Boas

$O(\lg \lg u)$ time means we binary search on bits of query word x.

$\text{binary}(x) = \underbrace{01100}_{high(x)}\underbrace{11100}_{low(x)} \in S$

low-order bits (low(x)) distinguish $\sqrt{u}$ consecutive items.
high-order bits (high(x)) distinguish $\sqrt{u}$ of these clusters.



We create $\sqrt{u} + 1$ recursive substructures.

- $\sqrt{u}$ substructures: $S[0], S[1], \ldots, S[\sqrt{u} - 1]$ representing partition of universe (above).
- A single substructure $S.summary$ of size $\sqrt{u}$.
  $S.summary[i] = 1 \Leftrightarrow S[i]$ is non empty.

INSERT(x, S)
1 INSERT$(low(x), S[high(x)])$
2 INSERT$(high(x), S.summary)$

Analysis: $T(u) = 2T(\sqrt{u}) + O(1) \Rightarrow T = \lg u$, need to get rid of the 2.

Fix: $\forall S$ we add

- $S.min$ = the minimum element of S.
  - not stored in an $S[i]$ recursively (but still counts as making S non empty).
  - $S.min$ = none $\Leftrightarrow$ S is empty.
- $S.max$ = the minimum element of S, unlike the min, this is also stored recursively.

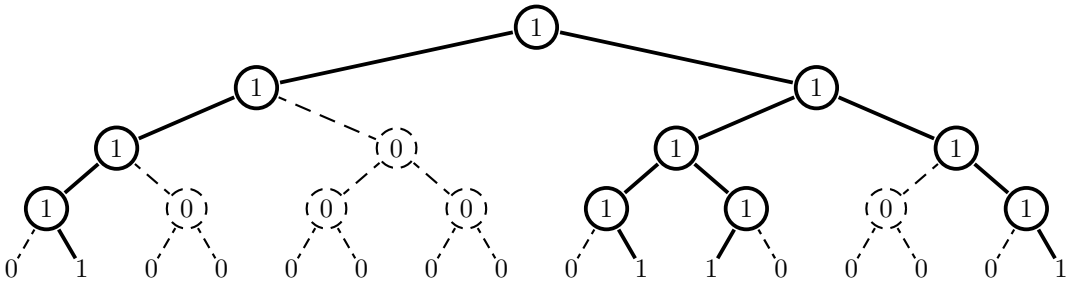
- An inserted item to $S[i]$ is either $\underbrace{\text{the only item in } S[i]}_{\text{set as S[i].min}}$ or $\underbrace{S.summary[i] \text{ was already 1}}_{\text{no need to update summary}}$.

INSERT(x, S)
```
 1  if S.min = none
 2     then S.min ← x  and  return
 3  if x < S.min
 4     then SWAP(x, S.min)
 5  if S[high(x)]  is empty  (.min = none)
 6     then S[high(x)].min ← low(x)
 7          INSERT(high(x), S.summary)
 8     else  INSERT(low(x), S[high(x)])
 9  if x > S.max
10     then S.max ← x
```

SUCCESSOR(x, S)
```
 1  if x < S.min
 2     then return S.min
 3  if low(x) < S[high(x)].max  (successor is in  S[high(x)])
 4     then return high(x) · √|S| + SUCCESSOR(low(x), S[high(x)])
 5     else  i ← SUCCESSOR(high(x), S.summary)
 6           return i · √|S| + S[i].min
```

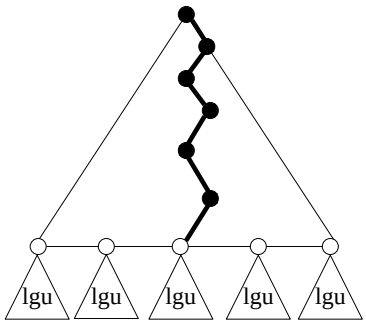
- $T(u) = T(\sqrt{u}) + O(1) \Rightarrow T = \lg \lg u$.
- predecessor and delete similar.
- only problem: space

y-fast tries



- store all prefixes of the binary representation of $x \in S$ in a hash table.
- binary search the root-to-x path to find the deepest 1-node (say y).
  - if $x \in left(y)$ then $\min(right(y)) = successor(x)$.
  - if $x \in right(y)$ then $\max(left(y)) = predecessor(x)$
- use a linked list on S to find the other succ/pred.
- $O(\lg \lg u)$ for pred/succ, $O(\lg u)$ for insert/delete, $O(n \lg u)$ space.

Indirection



- cluster elements of S into consecutive groups of size $\Theta(\lg u)$.
- store each group in balanced BST.
- use hash table solution on $\underbrace{representatives}_{\text{some } x \in \mathcal{U} \text{ in the right range}}$ from each group.

  $\Rightarrow O(n)$ space.

Predecessor/Successor:
- hash table finds pred/succ representatives.
- search the two corresponding BSTs.

Insert/Delete:

- insert/delete from BST (find via pred(x))

- if BST grows by $\times 2$ then split. if BST shrinks by $\times \frac{1}{4}$ then merge with neighbor & possibly resplit.
  $\Rightarrow$ takes $O(\lg u)$ time (why?) and happens only every $\Omega(\lg u)$ updates.
  $\Rightarrow O(1)$ amortized.

General Indirection:

Reduce any solution with $O(\lg \lg u)$ query time, $n \cdot (\lg u)^{O(1)}$ space, and $(\lg u)^{O(1)}$ update to $O(n)$ space and $O(\lg \lg u)$ update (query time unchanged).

References

[1] P. van Emde Boas, *Preserving Order in a Forest in less than Logarithmic Time*, FOCS, 75-84, 1975.

[2] Dan E. Willard, *Log-Logarithmic Worst-Case Range Queries are Possible in Space* $\Theta(n)$, Inf. Process. Lett. 17(2): 81-84 (1983)

[3] M. Fredman, D. E. Willard, *Surpassing the Information Theoretic Bound with Fusion Trees*, J. Comput. Syst. Sci, 47(3):424-436, 1993.

[4] A. Andersson, P. B. Miltersen, M. Thorup, *Fusion Trees can be Implemented with* AC^0 *Instructions Only*, Theor. Comput. Sci, 215(1-2): 337-344, 1999.