**Predecessor**

- Predecessor Problem
- van Emde Boas
- Tries

**Predecessors**

- **Predecessor problem.** Maintain a set $S \subseteq U = \{0, \ldots, u-1\}$ supporting
  - predecessor(x): return the largest element in $S$ that is $\leq x$.
  - successor(x): return the smallest element in $S$ that is $\geq x$.
  - insert(x): set $S = S \cup \{x\}$
  - delete(x): set $S = S \setminus \{x\}$

**Applications.**
- Simplest version of nearest neighbor problem.
- Several applications in other algorithms and data structures.
- Central problem for internet routing.
Routing IP-Packets

- Where should we forward the packet to?
- To address matching the longest prefix of 192.110.144.123.
- Equivalent to predecessor problem.

Best practical solutions based on advanced predecessor data structures [Degermark, Brodnik, Carlsson, Pink 1997]

### Predecessors

Predecessors

- Routing IP-Packets
  - Where should we forward the packet to?
  - To address matching the longest prefix of 192.110.144.123.
  - Equivalent to predecessor problem.
  - Best practical solutions based on advanced predecessor data structures [Degermark, Brodnik, Carlsson, Pink 1997]

### Predecessor

- Predecessor Problem
- van Emde Boas
  - Tries

### van Emde Boas

- Goal. Static predecessor with $O(\log \log u)$ query time.
- Solution in 5 steps.
  - Bitvector. Very slow
  - Two-level bitvector. Slow.
  - ...
  - van Emde Boas [Boas 1975]. Fast.
Solution 1: Bitvector

- Data structure. Bitvector.
- Predecessor(x): Walk left.
- Time. $O(u)$

Solution 2: Two-Level Bitvector

- Data structure. Top bitvector + $u^{1/2}$ bottom bitvectors.
- Predecessor(x): Walk left in bottom + walk left in top + walk left bottom.
  - To find indices in top and bottom write $x = h(x) \cdot u^{1/2} + l(x) = h(x) \cdot u^{1/2} + l(x)$
  - Index in top is $h(x)$ and index in bottom is $l(x)$.
- Time. $O(u^{1/2} + u^{1/2} + u^{1/2}) = O(u^{1/2})$

Solution 3: Two-Level Bitvector with less Walking

- Data structure. Solution 2 with min and max for each bottom structure.
- Predecessor(x):
  - If $h(x)$ in top and $l(x)$ ≥ min in bottom then walk left in bottom.
  - if $h(x)$ in top and $l(x)$ < min or $h(x)$ not in top walk left in top. Return max at first non-empty position in top.
- We either walk in bottom or top.
- Time. $O(u^{1/2})$
- Observation.
  - Query is walking left in one vector of size $u^{1/2} + O(1)$ extra work.
  - Why not walk using a predecessor data structure?

Solution 4: Two-Level Bitvector within Top and Bottom

- Data structure. Apply solution 3 to top and bottom structures of solution 3.
- Walking left in vector of size $u^{1/2}$ now takes $O((u^{1/2})^{1/2}) = O(u^{1/4})$ time.
- Each level adds $O(1)$ extra work.
- Time. $O(u^{1/4})$
- Why not do this recursively?
Solution 5: van Emde Boas

- Data structure. Apply recursively until size of vectors is constant.
- Time. \( T(u) = T(u^{1/2}) + O(1) = O(\log \log u) \)
- Space. \( O(u) \)

van Emde Boas

- Theorem. We can solve the static predecessor problem in
  - \( O(u) \) space.
  - \( O(\log \log u) \) time.
- Combined with perfect hashing we can reduce space to \( O(n) \) [Mehlhorn and Näher 1990].
- Easy to add insert and delete.

Predecessor

- Predecessor Problem
- van Emde Boas
- Tries

Tries

- Goal. Static predecessor with \( O(n) \) space and \( O(\log \log u) \) query time.
- Equivalent to van Emde Boas but different perspective. Simpler?
- Solution in 3 steps.
  - Trie. Slow and too much space.
  - X-fast trie. Fast but too much space.
  - Y-fast trie. Fast and little space.
Trie. Tree T of prefixes of binary representation of keys in S.
• Depth of T is $\log u$.
• Number of nodes in T is $O(n \log u)$.

Data structure.
• T as binary tree with min and max for each node + keys ordered in a linked list.
• Predecessor(x): Top-down traversal to find the longest common prefix of x with T.
  • x branches of T to right $\Rightarrow$ Predecessor(x) is max of sibling branch.
  • x branches of T to left $\Rightarrow$ Successor(x) is min of sibling branch. Use linked list to get predecessor(x).
• Time. $O(\log u)$
• Space. $O(n \log u)$

Solution 1: Top-down Traversal

Predecessor(x): Binary search over levels to find longest matching prefix with x.
• Example. Predecessor(9 = 1001$_2$):
  • 10$_2$ in $d_2$ exists $\Rightarrow$ continue in bottom 1/2 of tree.
  • 100$_2$ in $d_3$ exists $\Rightarrow$ continue in bottom 1/4 of tree.
  • 1001$_2$ in $d_4$ does not exist $\Rightarrow$ 100$_2$ is longest prefix.
• Time. $O(\log \log u)$

Solution 2: X-Fast Trie

Data structure.
• For each level store a dictionary of prefixes of keys + solution 1.
• Example. $d_1 = \{0, 1\}$, $d_2 = \{00, 10, 11\}$, $d_3 = \{000, 001, 100, 101, 111\}$, $d_4 = S$
• Space. $O(n \log u)$
Theorem. We can solve the static predecessor problem in $O(\log \log u)$ time and $O(n \log u)$ space.

How do we get linear space?

Solution 2: X-Fast Trie

- **Theorem.** We can solve the static predecessor problem in
  - $O(\log \log u)$ time
  - $O(n \log u)$ space.
  - How do we get linear space?

Solution 3: Y-Fast Trie

- **Theorem.** We can solve the static predecessor problem in $O(\log \log u)$ time and $O(n)$ space.
Y-Fast Tries

- **Theorem.** We can solve the static predecessor problem in
  - $O(n)$ space.
  - $O(\log \log u)$ time.

- **Theorem.** We can solve the dynamic predecessor problem in
  - $O(n)$ space
  - $O(\log \log u)$ expected time for predecessor and updates.

From dynamic hashing

Predecessor

- Predecessor Problem
- van Emde Boas
- Tries