On Dynamic Range Reporting in One Dimension

Christian Mortensen[1] Rasmus Pagh[1] Mihai Pătraşcu[2]

[1]IT U. Copenhagen

[2]MIT

STOC – May 22, 2005

Range Reporting in 1D

Maintain a set S, $|S| = n$, under:

INSERT(x): $S \leftarrow S \cup \{x\}$

DELETE(x): $S \leftarrow S \setminus \{x\}$

REPORT(a, b): return $S \cap [a, b]$

Range Reporting in 1D

Maintain a set S, $|S| = n$, under:

INSERT(x): $S \leftarrow S \cup \{x\}$

DELETE(x): $S \leftarrow S \setminus \{x\}$

REPORT(a, b): return $S \cap [a, b]$

Model: Word RAM, w-bit words

$S \subset \{0, \ldots, 2^w - 1\}$

Range Reporting in 1D

Maintain a set S, $|S| = n$, under:

INSERT(x): $S \leftarrow S \cup \{x\}$

DELETE(x): $S \leftarrow S \setminus \{x\}$

REPORT(a, b): return $S \cap [a, b]$

Alternative query

FINDANY(a, b): return any $y \in S \cap [a, b]$, or EMPTY

Updates maintain S in sorted order.
Then, just scan left or right starting with y.

Model: Word RAM, w-bit words

$$S \subset \{0, \ldots, 2^w - 1\}$$

Search Problems

Exact Search

$\text{MEMBER}(x)$: is $x \in S$?

Search Problems

Predecessor Search

$\text{PRED}(x)$: return $\max\{y \in S \mid y \leq x\}$

Exact Search

$\text{MEMBER}(x)$: is $x \in S$?

Predecessor Search

$\text{PRED}(x)$: return $\max\{y \in S \mid y \le x\}$

$\Downarrow \quad \text{PRED}(b)$

Range Reporting in 1D

$\text{FINDANY}(a, b)$: return any $y \in S \cap [a, b]$

$\Downarrow \quad \text{FINDANY}(x, x)$

Exact Search

$\text{MEMBER}(x)$: is $x \in S$?

Search Problems

Range Reporting in 2D

$\text{EMPTY}([a,b]\times[c,d])$: is $S\cap([a,b]\times[c,d])=\emptyset$?

$\Downarrow$ "colored predecessor problem"

Predecessor Search

$\text{PRED}(x)$: return $\max\{y\in S \mid y\leq x\}$

$\Downarrow$ $\text{PRED}(b)$

Range Reporting in 1D

$\text{FINDANY}(a,b)$: return any $y\in S\cap[a,b]$

$\Downarrow$ $\text{FINDANY}(x,x)$

Exact Search

$\text{MEMBER}(x)$: is $x\in S$?

Hardness of Range Reporting

Predecessor search:

$\Omega(\frac{\lg w}{\lg \lg w})$ per query, even statically

$O(\lg w)$ per query/update: van Emde Boas recursion.

Hardness of Range Reporting

Predecessor search:

$\Omega(\frac{\lg w}{\lg \lg w})$ per query, even statically

$O(\lg w)$ per query/update: van Emde Boas recursion.

Static range reporting $\mathcal{MAGIC}$:

- $O(1)$ query [MNSW – STOC'95]
- ... and $O(n)$ space [ABR – STOC'01]

Hardness of Range Reporting

Predecessor search:

$\Omega(\frac{\lg w}{\lg\lg w})$ per query, even statically

$O(\lg w)$ per query/update: van Emde Boas recursion.

Static range reporting $\mathcal{MAGIC}$:

- $O(1)$ query [MNSW – STOC'95]
- …and $O(n)$ space [ABR – STOC'01]

Dynamize these solutions $\Rightarrow$ tradeoff:

- $O(w^{\varepsilon})$ per update, $O(1)$ per query

$\updownarrow$

- $O(\lg w)$ per update, $O(\lg w)$ per query

Not so magical: converges to van Emde Boas.

We achieve:

- $O(\lg w)$ updates
- $O(\lg \lg w)$ queries
- $O(n)$ space

Dynamic $\mathcal{MAGIC}$

We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg \lg w)$ queries
- $O(n)$ space

Dynamic $\mathcal{MAGIC}$

We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg\lg w)$ queries
  exponential improvement over van Emde Boas
  in terms of universe size u, this is $O(\lg\lg\lg u)$
- $O(n)$ space

Dynamic $\mathcal{MAGIC}$

We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg \lg w)$ queries
  exponential improvement over van Emde Boas
  in terms of universe size u, this is $O(\lg \lg \lg u)$
- $O(n)$ space

$\mathcal{MAGICAL}$ ingredients:

Dynamic $\mathcal{MAGIC}$

We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg \lg w)$ queries
  exponential improvement over van Emde Boas
  in terms of universe size u, this is $O(\lg \lg \lg u)$
- $O(n)$ space

$\mathcal{MAGICAL}$ ingredients:

- eye of a newt

Dynamic $\mathcal{MAGIC}$

We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg\lg w)$ queries
  exponential improvement over van Emde Boas
  in terms of universe size u, this is $O(\lg\lg\lg u)$
- $O(n)$ space

$\mathcal{MAGICAL}$ ingredients:

- eye of a newt
- bat wing

Dynamic $\mathcal{MAGIC}$

We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg \lg w)$ queries
  exponential improvement over van Emde Boas
  in terms of universe size u, this is $O(\lg \lg \lg u)$
- $O(n)$ space

$\mathcal{MAGICAL}$ ingredients:

- eye of a newt
- bat wing
- new, subtle recursion idea

Dynamic $\mathcal{MAGIC}$

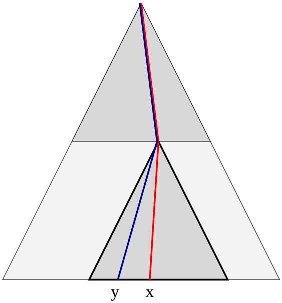
We achieve:

- $O(\lg w)$ updates
  need predecessor query if S is maintained sorted
- $O(\lg\lg w)$ queries
  exponential improvement over van Emde Boas
  in terms of universe size u, this is $O(\lg\lg\lg u)$
- $O(n)$ space

$\mathcal{MAGICAL}$ ingredients:

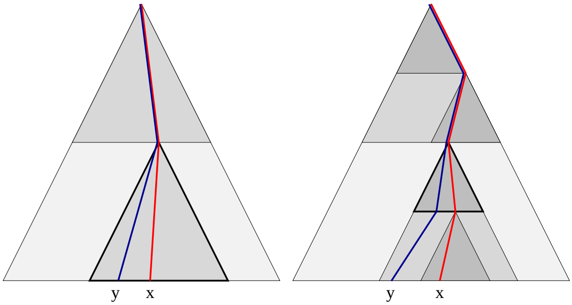
- eye of a newt
- bat wing
- new, subtle recursion idea
- dynamic perfect hashing in sublinear space

Van Emde Boas Recursion



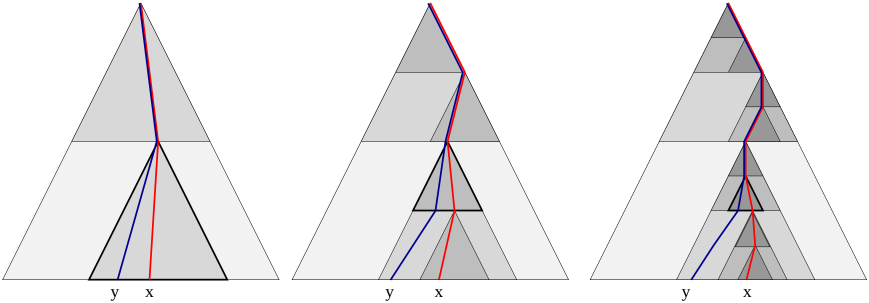
Binary search for longest common prefix of x and $\textsc{pred}(x)$.

Van Emde Boas Recursion



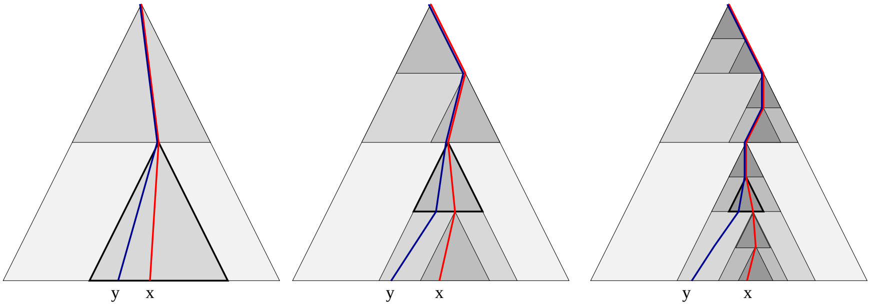
Binary search for longest common prefix of x and $\textsc{Pred}(x)$.

Van Emde Boas Recursion



Binary search for longest common prefix of x and $\textsc{pred}(x)$.

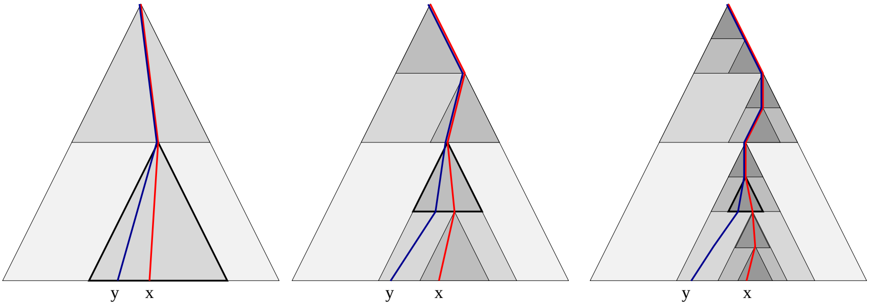
Van Emde Boas Recursion



Binary search for longest common prefix of x and $\text{PRED}(x)$.

Interpret integers at different levels of detail.

Van Emde Boas Recursion

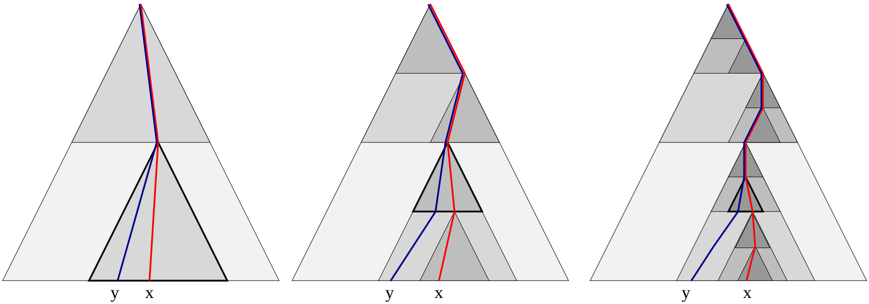


Binary search for longest common prefix of x and PRED(x).

Interpret integers at different levels of detail.

- van Emde Boas examines levels of detail sequentially

Van Emde Boas Recursion

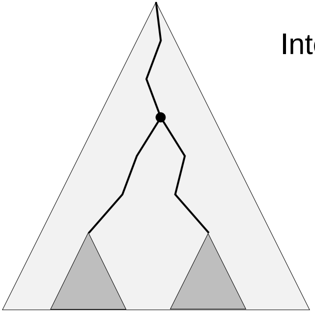


Binary search for longest common prefix of x and $\text{PRED}(x)$.

Interpret integers at different levels of detail.

- van Emde Boas examines levels of detail sequentially
- we do a binary search on the levels of detail

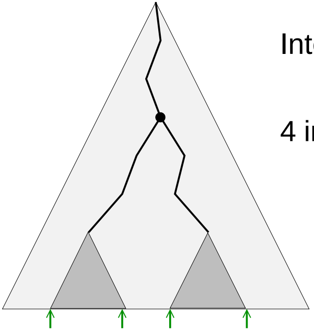
What are we searching for?



Interpret S as paths in trie of height w

$\Rightarrow n - 1$ branching nodes

What are we searching for?

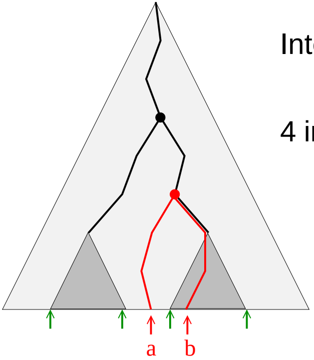


Interpret S as paths in trie of height w

$\Rightarrow n - 1$ branching nodes

4 interesting values per branching node

What are we searching for?



Interpret S as paths in trie of height w
$\Rightarrow n - 1$ branching nodes

4 interesting values per branching node

FINDANY(a, b)

- compute LCA(a, b)
- find lowest branching ancestor of the LCA
- check if any extreme point is in $[a, b]$

Lowest branching ancestor?

Trouble: finding the lowest branching ancestor of arbitrary v is as hard as predecessor search

Lowest branching ancestor?

Trouble: finding the lowest branching ancestor of arbitrary v is as hard as predecessor search

But we don't always need to find it

Lowest branching ancestor?

Trouble: finding the lowest branching ancestor of arbitrary v is as hard as predecessor search

But we don't always need to find it

Assume v is on an active path:

- if true, find ancestor faster
- if false, fail

Lowest branching ancestor?

Trouble: finding the lowest branching ancestor of arbitrary v is as hard as predecessor search

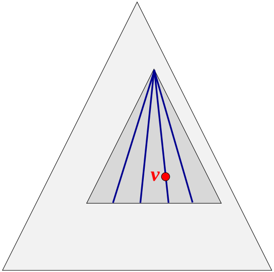
But we don't always need to find it

Assume v is on an active path:

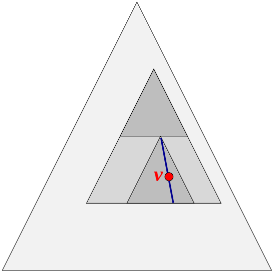
- if true, find ancestor faster
- if false, fail

Happens only when $S \cap [a, b] = \emptyset$
$\Rightarrow$ witness verification catches the error

Binary search for level L such that:



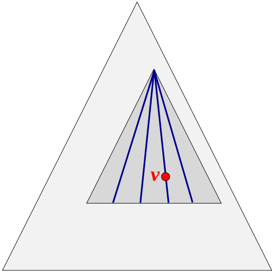
Level $L - 1$: branching



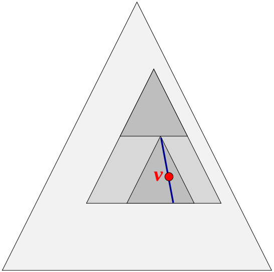
Level L: no branching

Binary search on levels of detail

Binary search for level L such that: $O(\lg \lg w)$



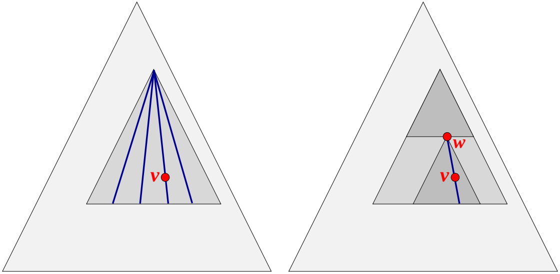
Level $L - 1$: branching



Level L: no branching

Binary search on levels of detail

Binary search for level L such that: $O(\lg\lg w)$



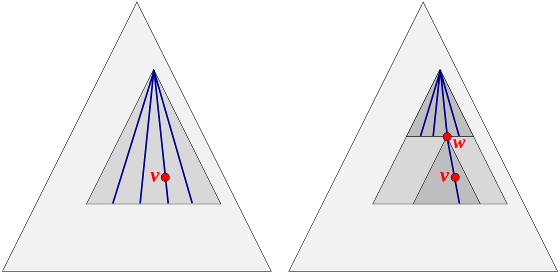
Level $L-1$: branching Level L: no branching

Then, w holds pointer to lowest branching ancestor.

Nice, but can we update?

Binary search on levels of detail

Binary search for level L such that: $O(\lg \lg w)$



Level $L - 1$: branching Level L: no branching

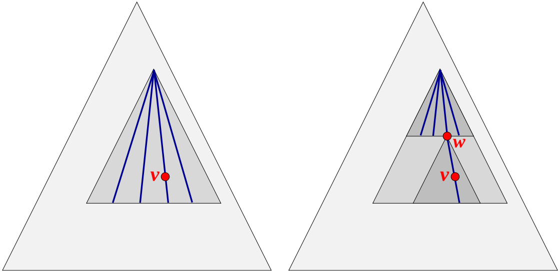
Then, w holds pointer to lowest branching ancestor.

Nice, but can we update?

- at level L, must have branching immediately above

Binary search on levels of detail

Binary search for level L such that: $O(\lg \lg w)$



Level $L-1$: branching   Level L: no branching

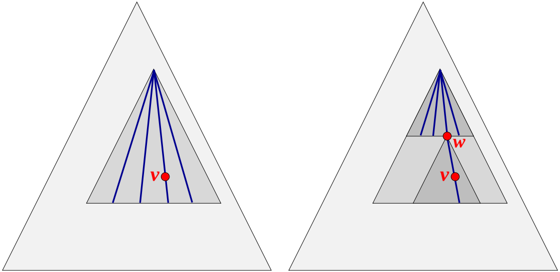
Then, w holds pointer to lowest branching ancestor.

Nice, but can we update?

- at level L, must have branching immediately above
- so ancestor in same "triangle" as w on level L

Binary search on levels of detail

Binary search for level L such that: $O(\lg \lg w)$



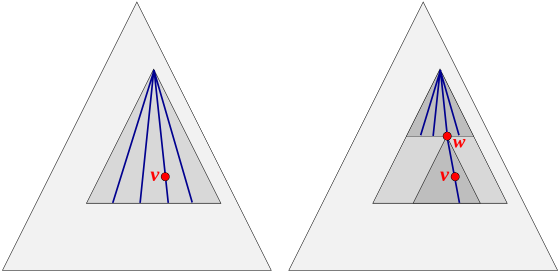
Level $L-1$: branching　　Level L: no branching

Then, w holds pointer to lowest branching ancestor.
Nice, but can we update?

- at level L, must have branching immediately above
- so ancestor in same "triangle" as w on level L
- for each branching node, ≤ 2 pointers to it at every level

Binary search on levels of detail

Binary search for level L such that: $O(\lg \lg w)$



Level $L-1$: branching   Level L: no branching

Then, w holds pointer to lowest branching ancestor.

Nice, but can we update? $O(\lg w)$

- at level L, must have branching immediately above
- so ancestor in same "triangle" as w on level L
- for each branching node, ≤ 2 pointers to it at every level

Achieving $O(n)$ space

Space: $O(\lg w)$ pointers to $O(n)$ branching nodes

But can encode as level pointers: $O(\lg w)$ bits each

$\Rightarrow O(n \lg^2 w)$ bits of “real information” $= o(n)$ words.

Achieving $O(n)$ space

Space: $O(\lg w)$ pointers to $O(n)$ branching nodes
But can encode as level pointers: $O(\lg w)$ bits each
$\Rightarrow O(n \lg^2 w)$ bits of "real information" $= o(n)$ words.

Bloomier filters:

- store vector $A[1 \,..\, U]$ of r-bit values
- only $N \ll U$ nonzero positions

Achieving $O(n)$ space

Space: $O(\lg w)$ pointers to $O(n)$ branching nodes
But can encode as level pointers: $O(\lg w)$ bits each
$\Rightarrow O(n \lg^2 w)$ bits of "real information" $= o(n)$ words.

Bloomier filters:

- store vector $A[1..U]$ of r-bit values
- only $N \ll U$ nonzero positions

$$N = O(n \lg w),\ U = O(2^w),\ r = O(\lg w)$$

Achieving $O(n)$ space

Space: $O(\lg w)$ pointers to $O(n)$ branching nodes
But can encode as level pointers: $O(\lg w)$ bits each
$\Rightarrow O(n \lg^2 w)$ bits of "real information" $= o(n)$ words.

Bloomier filters:

- store vector $A[1..U]$ of r-bit values
- only $N \ll U$ nonzero positions

$$N = O(n \lg w),\ U = O(2^w),\ r = O(\lg w)$$

Space lower bound: $\sim N(r + \lg U)$ bits $\sim N$ words.

Achieving $O(n)$ space

Space: $O(\lg w)$ pointers to $O(n)$ branching nodes
But can encode as level pointers: $O(\lg w)$ bits each
$\Rightarrow O(n \lg^2 w)$ bits of "real information" $= o(n)$ words.

Bloomier filters:

- store vector $A[1 \,..\, U]$ of r-bit values
- only $N \ll U$ nonzero positions

$$N = O(n \lg w),\ U = O(2^w),\ r = O(\lg w)$$

Space lower bound: $\sim N(r + \lg U)$ bits $\sim N$ words.

Allow one-sided error:

- if $A[i] \neq 0$, answer must be correct
- if $A[i] = 0$, answer can be wrong

Low-Space Structures

First sublinear-space solution to dynamic Bloomier filters:
$O(n(r + \lg \lg u))$ bits

Via first sublinear-space solution to dynamic perfect hashing

We prove matching lower bounds
improves [CKRT – SODA'04]

Story Time

Almost done. . .

Story Time

$\mathcal{THE\ END}$