# Lower Bounds for Predecessor Search

Mihai Pătraşcu
at&t

MADALGO Summer School 2010
Monday, Afternoon I

## Results

[van Emde Boas FOCS'75]
Predecessor search in O($n$) space, O(lglg $u$) time

[Pătraşcu, Thorup STOC'06] Let $n^{1+\varepsilon} \le u \le n^{\text{polylog}(n)}$
Any data structure of size $n \cdot \log^{O(1)} n$ requires time $\Omega(\lg\lg u)$

In fact, n/u/space trade-off completely understood ☺

$$\min \begin{cases} \log_w n \\ \lg \frac{\ell - \lg n}{a} \\ \frac{\lg \frac{\ell}{a}}{\lg\left(\frac{a}{\lg n} \cdot \lg \frac{\ell}{a}\right)} \\ \frac{\lg \frac{\ell}{a}}{\lg\left(\lg \frac{\ell}{a} \Big/ \lg \frac{\lg n}{a}\right)} \end{cases}$$

## Colored Predecessor

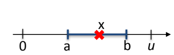
Color the set red-blue $S = \{ x_1, x_2, x_3, x_4, x_5, \dots \}$
Lower bound holds even for finding <u>color</u> of predecessor
⇒ colored predecessor ≡ predecessor
⇒ existential static 1D stabbing ≡ predecessor

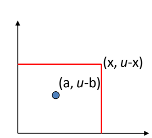


Query stabs a segment ⇔ predecessor is blue

## Colored Predecessor

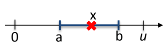
Color the set red-blue $S = \{ x_1, x_2, x_3, x_4, x_5, \dots \}$
Lower bound holds even for finding <u>color</u> of predecessor
⇒ colored predecessor ≡ predecessor
⇒ existential static 1D stabbing ≡ predecessor
⇒ static existential 2D dominance ≡ predecessor

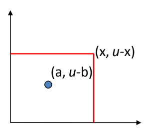


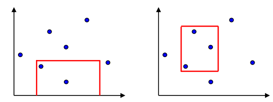
## Colored Predecessor

In general: stabbing in $d$ dimensions
reduces to dominance queries in $2d$ dimensions




## Review: 2D Range Queries



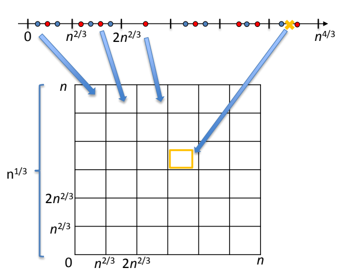
| | 3-sided | 4-sided |
|---|---|---|
| Rank space: | $O(1)$ | $O(\lg\lg n)$ |
| Large universe: | $\Theta(\lg\lg u)$ | $\Theta(\lg\lg u)$ |

Also tight!

Colored predecessor

## 4-sided Queries in Rank Space

Predecessor search for $u=n^{4/3}$ requires time $\Omega(\lg\lg n)$



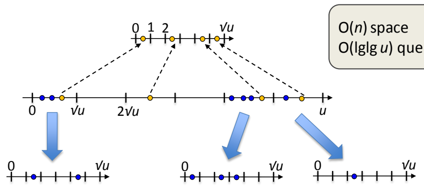

Time for…

## The Lower Bound Proof

Let $u = n^3$
Any data structure of size $O(n)$ requires time $\Omega(\lg\lg n)$

## Review: van Emde Boas

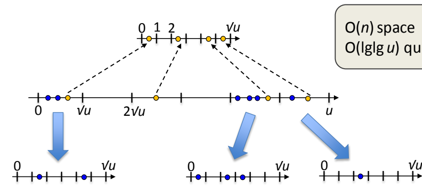
pred($q$, $S$): **if** $\lfloor q/\sqrt{u} \rfloor \in$ hash table,
**return** pred($q$ mod $\sqrt{u}$, bottom structure)
**else return** pred($\lfloor q/\sqrt{u} \rfloor$, top structure)




O($n$) space
O(lglg $u$) query

## Review: van Emde Boas

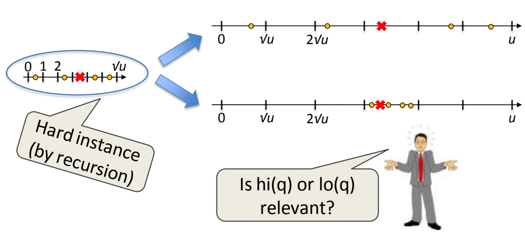
pred($q$, $S$): **if** hi(q) $\in$ hash table,
**return** pred(lo(q), bottom structure)
**else return** pred(hi(q), top structure)




O($n$) space
O(lglg $u$) query

## The Hard Instance

Intuition: In one memory probe,
one can at most reduce the universe $u \mapsto \sqrt{u}$

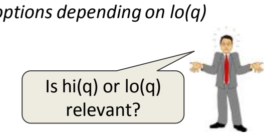



## The First Memory Access

van Emde Boas always tries top first ⇒ read address $f$(hi(q))
Can easily be fooled (put all elements in a $\sqrt{u}$ interval)

*But I can access O(n) memory locations!*

*I'll choose among O($\sqrt{n}$) options depending hi(q)*
*and O($\sqrt{n}$) options depending on lo(q)*



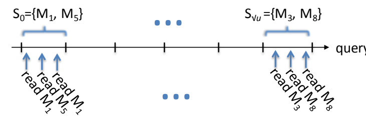
## Updating our Intuition

Simple intuition: In one memory probe, one can at most reduce the universe $u \mapsto \sqrt{u}$

Refined intuition:

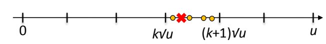
- either pred(hi(q), hi(S)) or pred(lo(q), lo(S)) is relevant
- query algorithm can read $\leq\sqrt{n}$ cells as f(relevant part)

## Formally

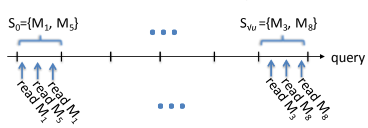



If $(\exists)k \quad |S_k| \leq \sqrt{n}$:

- place query & data set in segment $k$
- $1^{st}$ memory access = $f(\text{lo}(q)) \in S_k$

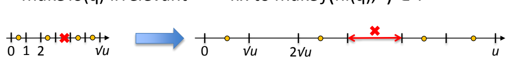



## Formally




Otherwise $(\forall)k \quad |S_k| \geq \sqrt{n}$:

- choose T = { $O(\sqrt{n} \cdot \lg n)$ cells } $\Rightarrow$ each $S_k$ is hit
- $1^{st}$ memory access = $f(\text{hi}(q), \text{lo}(q)) \in S_{\text{lo}(q)}$
- make lo(q) irrelevant $\Rightarrow$ fix to make $f(\text{hi}(q),*) \in T$




## What Did We Prove?

If there exists a solution to Pred($n$, $u$) with:

- space complexity: $O(n)$
- query complexity: $t$ memory reads

$\Rightarrow$

There exists a solution to Pred($n$, $\sqrt{u}$) with:

- space complexity: $O(n)$
- $O(\sqrt{n} \cdot \lg n)$ "public cells"
- query complexity: $t$-1 memory reads

... can be read free of charge

## Induction?

**Induction:**

Pred($n$,$u$) ↦ Pred($n$,√$u$) ↦ Pred($n$,$u^{1/4}$) ↦ ... ↦ Pred($n$,$u^{1/t}$)

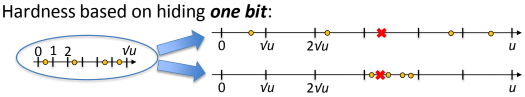
time $t$ ↦ time $t$-1 ↦ time $t$-2 ↦ ... ↦ time 0

**Base case:** If $u^{1/t} \geq 2$, cannot solve with no memory access! *

* Except o($n$) public bits.

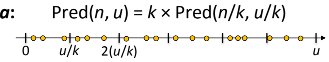
## Dealing with Public Bits

Hardness based on hiding ***one bit***:



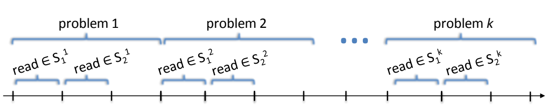

In 2nd round, there are O(√$n$ · lg²$n$) public bits about the input!

***New idea***: Pred($n$, $u$) = $k$ × Pred($n/k$, $u/k$)




$k \gg \sqrt{n} \cdot \lg^2 n$ ⇒ With O(√$n$ · lg²$n$) public bits, most sub-problems are still hard.

## New Induction Plan




***Main Lemma***: Fix algorithm to read from a set of $(nk)^{1/2}$ cells

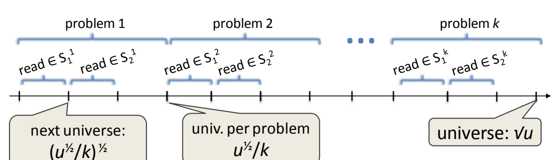
"Proof":

- problem j is nice if (∃)$\alpha$: $|S_\alpha^j| \leq (n/k)^{1/2}$
  ⇒ fix hi-part in problem j to $\alpha$
- problem j is not nice if (∀)$\alpha$: $|S_\alpha^j| > (n/k)^{1/2}$
  ⇒ choose T to hit all such $S_\alpha^j$

$k \cdot (n/k)^{1/2} = (nk)^{1/2}$

$|T| \approx n / (n/k)^{1/2} = (nk)^{1/2}$

## New Induction Plan




Set $u$=$n^3$ and $k$ = $n^{3/4}$

Ω(lglg $n$) times ☺

| | Time | Public bits |
|---|---|---|
| Pred($n$, $n^3$) | $t$ | 0 |
| ↦ Pred($n$, $n^{1.5}$) ↦ $n^{3/4}$ × Pred($n^{1/4}$, $n^{3/4}$) | $t$-1 | √$n$ |
| ↦ $n^{3/4}$ × Pred($n^{1/4}$, $n^{3/8}$) ↦ ••• | $t$-2 | $n^{7/8}$ |

## Main Lemma: "Proof" ↦ Proof

***Main Lemma***: Fix algorithm to read from a set of $(nk)^{½}$ cells

- problem j is nice if $(\exists)\alpha$: $|S_\alpha^j| \le (n/k)^{½}$
  - ⇒ fix hi-part in problem j to $\alpha$
- problem j is not nice if $(\forall)\alpha$: $|S_\alpha^j| > (n/k)^{½}$
  - ⇒ choose T to hit all such $S_\alpha^j$

But: Public bits = $f$(database)
$1^{st}$ cell read by query = $f$(public bits)

Look at what query does ⇒ Fix some problem adversarially
⇒ public bits change ⇒ Query does something else!

## Main Lemma: "Proof" ↦ Proof

***New claim.*** We can publish $(nk)^{½}$ cells such that:
Pr[random query reads a published cell] ≥ 1/100

Induction: If initial query time < (lglg $u$)/100
⇒ at the end E[query time] < 0 ⇒ contradiction

Proof:

But why does is work?

- Publish random sample T = { $(nk)^{½}$ cells }
- For each problem j where lo(q) is relevant (fixed hi=α)
  publish $S_\alpha^j$ only if $|S_\alpha^j| \le (n/k)^{½}$

## An Encoding Argument

*Impossible*: Encode A[1..$k$] ∈ $\{0,1\}^k$ with less than $k$ bits on avg.

Proof by contradiction:

- Assume Pr[random query reads a published cell] < 1/100
- Use this data structure to create an impossible encoder.

## The Encoding

1. Generate one random query/subproblem $(q_1, q_2, \ldots, q_k)$
2. Generate random database depending on queries:
   A[j]=0 ⇒ lo($q_j$) is relevant in problem j
   A[j]=1 ⇒ hi($q_j$) is relevant in problem j
3. Write public bits = $f$(database) **→ o(k) bits**
4. Classify queries: when $|S_{hi(qj)}^j| \le (n/k)^{½}$ query is ☺ iff A[j]=0
   when $|S_{hi(qj)}^j| > (n/k)^{½}$ query is ☺ iff A[j]=1
5. By assumption, E[number of ☹ queries] ≥ 99% $k$
   So decoder can learn 99% of A[1..$k$] from public bits!
6. Write in encoding which guesses are wrong
   **→ lg ($k$ choose $k$/100) ≪ $k$ bits**