6.851

André Schulz' Notes

Lecture 9

Models of Computation

Cell-probe-Model:

- Memory cells have size w ← Parameter of the model
- We can read/write in cells ~~for free~~ at cost of 1
- any computation is free
- not practical but good model to prove lower bounds

Transdichotomous RAM:

- memory cells have size w (and we assume $w \geq \log n$)
- each operation can modify only $O(1)$ cells
- cells can be addressed arbitrarily (RAM)

Variant 1 (Word RAM): $O(1)$ operations:

+, -, *, \, %, &, |, ^, ~, >>, <<
(C-Style)

Variant 2 (AC^0-RAM): $O(1)$ operations:
have to be realized by an AC^0 circuit
(constant depth, poly. size in w)
No multiplications !
(Rest of word-RAM ops are fine)

Pointer Machine Model: data structure is modeled as directed graph with bounded degree

BST Model: graph is a binary search tree

Cell Probe
Transdichotomous RAM: AC^0 RAM, word-RAM
Pointer Machine
BST Model

Stronger Model

Fixed universe assumption: universe is the set of integers from $\{0, \dots, u-1\}$

Successor/Predecesor in fixed universe

Overview (classic results)

DS	time/op	Space	model
balanced trees	$O(\log n)$	$O(n)$	BST
van Emde Boas-Trees	$O(\log\log u)$	$O(u)$	word/AC^0 RAM [van Emde Boas '75]
y-Fast trees	$O(\log\log u)$ w.h.p.	$O(n)$	word RAM [Willard '83]
Fusion Trees	$O(\log n / \log\log u)$	$O(n)$	word RAM [Willard Fredman '9] AC^0 RAM [Andersen, Miltersen, Thorup '99]

van Emde Boas Trees

- represent the set $S \subseteq \mathcal{U}$ as bit vektor

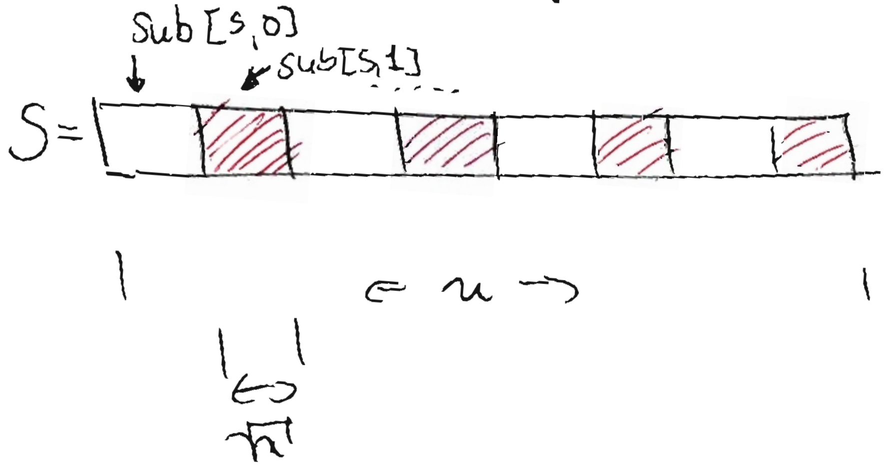
$\mathcal{U} = \{0, 1, \dots, 7\}$ $\quad S_1 = \{1, 3, 5, 6, 7\}$

$\Rightarrow$

0	1	2	3	4	5	6	7
	1		1		1	1	1

The operations insert, find, delete, succ, pred should all need only $O(\log\log u)$ time

Idea: Use recursion



bitvektor sliced in $\sqrt{u}$ Blocks

We store the information if the blocks are empty in an auxiliary structure Summary[S]

Summary = Bitvector of size $\sqrt{u}$

High x = block of x
Low x = position of x inside High x

1st approach: successor(S, x)

k = Successor(Sub[S, high x], low x)
if $k < \infty$ report $k + \text{high } x \cdot \sqrt{|S|}$
else z = successor(Summary[S], high x)
if $z == \infty$ return ∞
else ~~return~~ y = Successor(Sub[S, z], $-\infty$)
return $z \cdot \sqrt{S} + y$

→ needs $T(u) = \underline{\underline{3}} \cdot T(\sqrt{u}) + O(1)$ time

too much! We have to get rid of the 3!

2nd Approach: Store for every substructure S also min S, max S

$\boxed{\text{Successor}(S, x)}$

if low x < max [sub [S], high x] — ∃ successor in subblock

then $j \leftarrow$ successor (~~low~~ sub[S, high x], low x)

return $\text{high}(x) \cdot \sqrt{|S|} + j$

else $k \leftarrow$ successor (summary [S], high x) — ∄ successor in subbloc

return $\min[\text{sub}[S, k]] + k \cdot \sqrt{S}$

Notice we have to update min/max S while inserting / deleting.

$\boxed{\text{INSERT}(S, x)}$

if min[S] = ∅ **then** min [S] = x — update min [S]

if x < min [S] **swap** x, min [S]

if min [SUB (S, high x)] = ∅ — 1st element in SUB[S, high

then Min [SUB [S, high x]] = low x

INSERT (SUMMARY [S], high (x))

else INSERT (SUB [S, high x], low x)) — not 1st element in SUB [S, hig

if x > MAX [S] **then** MAX [S] = X — update max [S]

van Emde Boas Tree need O(n) space!

(Homework)

X-fast Trees

bitvector for $S \subseteq U$

Successor/predecessor query: (1) go up until you reach "1"

(2) go down as close as possible

- this gives predecessor or successor, link all leaves to allow to answer all queries
- Store in every node min/max to speed up (2)
- the first "1" we find is the longest prefix for binary(x) in $\{ \text{binary}(y) \mid y \in S \}$
- Store all prefixes in Hashtable (perfect, dynamic) $O(1)$/opn w.h.p.

⇒ We have stored n "Paths" of length $O(\log n)$

↳ $O(n \cdot \log n)$ space (too much!)

⇒ Accessing Successor / Predecessor

↳ Binary search over Prefix length

$O(\log \log n)$

⇒ updates: we might add $O(\log n)$ prefixes

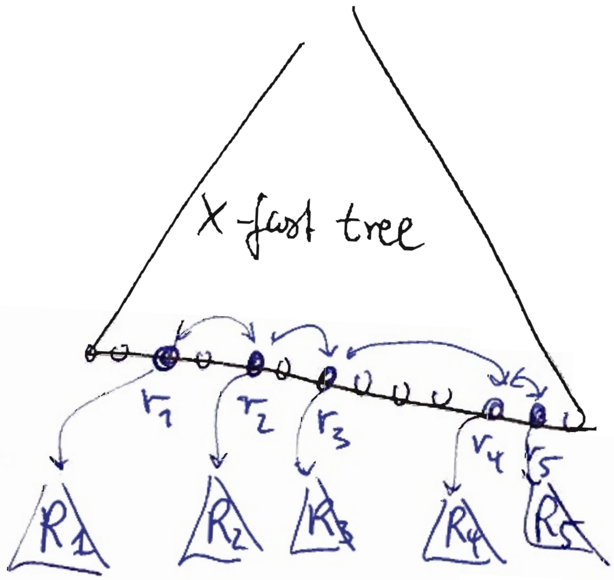
⇒ $O(\log n)$ (too much!)

y-fast-trees

(store the representatives in an X-fast-tree)

$|\{r_i \mid i \le k\}| = O\left(\frac{n}{\lg n}\right)$

$\Rightarrow$ space X-fast-tree : $O(n)$



all these trees need $O(\log n)$ space

$\Rightarrow O(n)$ all together

Every representative has a pointer to its BST

<u>Predecessor queries</u> :
- Find the predecessor in $\{r_1, r_2 \dots r_k\}$ say r_j $(O(\log\log n))$
- search in R_j & R_{j+1} $(O(\log\log n))$

<u>Updates</u> :
- update only in R_i $(O(\log\log n))$
- unless $\frac{\log n}{2} > |S_i|$ or $2 \log n < |S_i|$ (*)
  - $\downarrow$ merge 2 R_i's $(O(\log n))$
  - $\downarrow$ split R_i $O(\log(n))$

+ update x fast-tree $O(\log n)$

- (*) happens only every $O(\log n)$ opns

$\Rightarrow O(\log\log n)$ amortized !

→ $|\text{binary}(x)| \doteq w = \log u$

→ search for the longest prefix (binary search)

<u>$\log \log u$</u> !

→ drawbacks: insert/delete : $O(\log u)$
space : $O(n \log u)$

<u>Improvement</u> <u>y-fast trees</u> [Willard '83]

<u>Idea:</u> we group small subsets of our input together and store them in an additional DS (INDIRECTION)

• cluster size : $\Theta(\log u)$ $S_1 \dot\cup S_2 \dot\cup S_3 \dots S_k$ = clusters

• each cluster S_i is stored in a balanced BST (e.g. (2,4)-trees) R_i

$s_i \in S_i$
$s_j \in S_j$
$i > j$
$\Rightarrow s_i > s_j$

• for ever cluster S_i we have a representative r_i : $\forall s_i \in S_i \; \forall s_{i+1} \in S_{i+1}$

$$s_i \geq r_i > s_{i+1}$$