CS 840 **Winter 2013**

Introduction

Time & Place: 10:30-11:50 pm Wednesdays and Fridays.
Background
Requirements: somewhat modified from original handout

assignments/short paper review (3)	~45
project – paper review/start	~10
- written	~25
- present	~10
participation	~10

Starting:

Problem: Insert/Delete/find/find next largest etc. from ordered set

Solution: Balanced Search Tree (AVL, etc.)
Cost $\Theta(\lg n)$ time
linear number of "words"
can we do better?

Decision tree model:

Theorem: If n leaves then height $\geq [\lg n]$
include average path to leaf $\geq \lg n$

Counter Example: But Hashing takes O(1) on average for <u>static</u> can be made worst case. Indeed we can come up with a hashing approach that gives:
Worst case search time = O(1)
Expected update time = O(1)
The approach is "dynamic perfect hashing"
M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan, Dynamic perfect hashing: upper and lower bounds, SIAM J. Comput. 23 (1994), 738--761. http://citeseer.ist.psu.edu/dietzfelbinger90dynamic.html

Contradiction? No different model.

<u>To think out of the Box – first Define the Box!</u>

Now can we beat O(lg n) for insert/delete/successor
(if element not present find the next larger and call it successor)

In "general" – keys "real numbers" only have natural order
… not in worst case …

though for static: Interpolation search gives O(lglg n) extended search.

[- can we make it dynamic – mostly Mehlhorn and Tsakalidis (*JACM* 1993)]

Key idea of interpolation:

Guess position ((x – min-key) / (max-key – min-key) ∗ range-size) as location of value.

How far off will we be on average? – about $\sqrt{n}$

Why? Flip coin n times exp [#heads - #tails] = $\Theta(\sqrt{n})$.

So $T(n) = 1 + T(\sqrt{n})$.

How many times do we recurse

n lg n bits
$\sqrt{n}$ 1/2 lg n bits
…
lglg n times

Under what circumstances can we guarantee better than Θ(lg n) insert/delete/find closest?

A classic (and there are not many classics) data structure due to van Emde Boas in *SWAT* (now called *FOCS*) 1975 – then *Math. Syst. Theory* (1977).

Papers:
The version everyone remembers: Peter van Emde Boas: Preserving Order in a Forest in less than Logarithmic Time. FOCS 1975: 75-84
The polished journal form:
Peter van Emde Boas, R. Kaas, E. Zijlstra: Design and Implementation of an Efficient Priority Queue. Mathematical Systems Theory 10: 99-127 (1977)

But more general | insert
delete
search
successor
find closest above/below
& extract/find max/mix

I'll give a different presentation.

The result:

Universe of u values {0,…,u-1}
n of which are currently in the set.
The result lglg u time

lglg u – We have already seen how this bound arises

$$T(u) = T(\sqrt{u}) + 0(1)$$

or

$$\begin{aligned} T'(\lg u) &= T'((\lg u)/2) + 0(1) \\ &= \Theta(\lg\lg u) \end{aligned}$$

Start … we admit (at least for now) space depending on u.

Natural Start

Bit vector: u = 16, n = 4, S = {1, 9, 10, 15}

0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Insert/Delete/Find – O(1)
Successor/Pred – O(u)

Let's try to improve.

Carve into widgets … of size $\sqrt{u}$ (we need a $\sqrt{u}$ somehow)

00	01	10	11
w_0	w_1	w_2	w_3
0 1 0 0	0 0 0 0	0 1 1 0	0 0 0 1

w_i denotes $i\sqrt{u} + j$, $j = 0\ldots\sqrt{u} - 1$

so given a value $x = 9 = 1\ 0\ 0\ 1$ we have

high(x) = 10 (= 2)
low(x) = 01 (= 1)

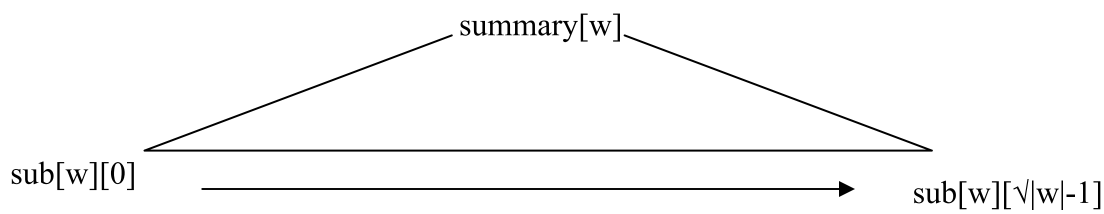
So we got to widget $w_2 = 0\ 1\ 1\ 0$, and look at element 1 which is 1.

Insert/Delete/Find cost is 0(1).

successor - $0(\sqrt{u})$ → look in appropriate widget → find next
→ find next non empty widget
→ find 1st in it

Revelation: each is a recursive call.

Representing widgets of size u as



So to insert

Insert(x,w)
    if sub[w][high(x)] is empty
        then Insert(high(x),summary(w))
    Insert(low(x),sub[w][high(x)])

Runtime: $T(u) = 2\,T(\sqrt{u}) + O(1)$

$= O(\lg u)$

Successor (x,w)

1. $j \leftarrow$ Successor (low(x), sub[w][high(x)])
   if $j < \infty$ then return $\text{high}(x)\sqrt{|w|} + j$
2. else $i \leftarrow$ Successor (high(x),summary[w])
3. $j \leftarrow$ Successor ($-\infty$, sub[w][i])
   return $i\sqrt{|w|} + j$

Runtime: 3 recursive calls.
Looks like $O((\lg u)^{\lg 3})$ [$\lg 3 = 1.58\ldots$]

Get down to 1 call in each

1 call $O(\lg \lg u)$
2 calls $O(\lg u)$
3 calls $O((\lg u)^{\lg 3})$

Fixes

If x has successor in sub[w][high(x)] then only 1 call

Otherwise 3 calls.

- So in each widget keep max value hence we don't make call 1 unless we need it for answer

So max 2 calls

- Now assume not in 1st call, so that's O(1)
- 2nd call gets appropriate sub widget (a real call)
- 3rd call just needs min value in sub widget (keep min, like max)

Hence only 1 "real" recursive call for successor

For Insert much the same but …

```
Insert(x,w)
    if  sub[w][high(x)] is empty
        then  Insert(high(x),summary[w])
    Insert(low(x),sub[w][high(x)]
    if  x < min[w]  then  min[w] ← x
    if  x > min[w]  then  max[w] ← x
```

2 recursive calls

⇒ Improvement if only 1 or 2 elements in a widget just store max/min

Now only 1 recursive call.

Runtime: $\Theta(\lg \lg u)$

Deletion: similar.

Further

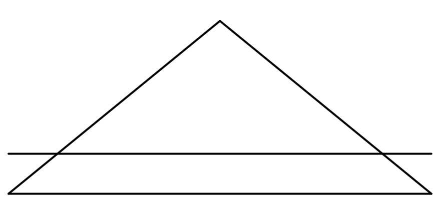
Lower bound Mehlhorn and Naher *SICOMP '88*

But Brodnik, Carlsson, Karlsson, Munro

O(1) time on Rambo (Random Access Memory with Byte Overlap ..a different model of memory in which words may share bits)

Another issue – space

We take O(u) ~ looks like $u \lg u$ bits but easy to reduce to u



⇐ lawnmower on $\lg \lg u$ levels

so structure on $u/\lg u$ objects
each is a bit vector length $\lg u$

How do we reduce this to something in n? Answer ~ hash