Dynamic order maintenance

Task: store a collection of items in a given order.

operations:

insert(x,y) → insert item y immediately after x (insert(ε, y) → insert at front)

delete(x) → delete item x

order(x,y) → return $\begin{cases} \text{true} & \text{if } x \text{ is before } y \\ \text{false} & \text{otherwise} \end{cases}$

e.g. insert(ε,a) : a

insert(a,b) a, b

insert(b,c) a, b, c

insert(a,d) a, d, b, c

order(a,c) → true

order(b,d) → false

Idea ①: linked list n = # stored items

$a \to d \to b \to c$

insert $O(1)$

delete $O(1)$

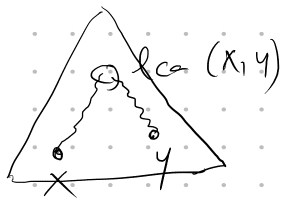
order $O(n)$

Idea ② balanced BST

insert(x,y) $O(\log n)$? $O(1)$

delete(x) $O(\log n)$? $O(1)$

order(x,y) $O(\log n)$ augmented trees
select/rank



Idea ③ store items as linked list, also store a label for each item, labels will correspond to order.

e.g. $\overset{1}{a} \to \overset{2}{b} \to \overset{3}{c}$

order(x,y): compare labels

insert(a,d)

$\overset{1}{a} \to \overset{1.5}{d} \to \overset{2}{b} \to \overset{3}{c}$

insert(a,e)

$\overset{1}{a} \to \overset{1.25}{e} \to \overset{1.5}{d} \to \overset{2}{b} \to \overset{3}{c}$

problem: need too many bits for labels

new strategy: use integer labels, leave some gaps between labels.
Want labels to be $\leq n^c$, so only $O(\log n)$ bits needed.

e.g.

$$\overset{10}{a} \longrightarrow \overset{20}{b} \longrightarrow \overset{30}{c}$$

insert(a, d)

$$\overset{10}{a} \longrightarrow \overset{15}{d} \longrightarrow \overset{20}{b} \longrightarrow \overset{30}{c}$$

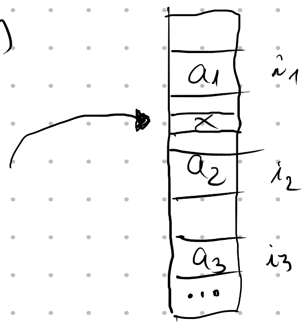
If no place to insert new item,
then we "loosen up" list by relabeling some items.

Goal: • $O(1)$ time for order(x,y)
• $O(\log n)$ amortized time for updates

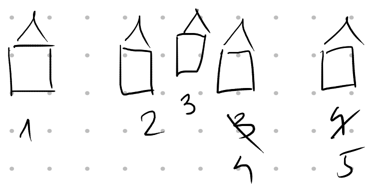
(also known as: "list labeling problem")

Applications: 1) items stored in memory

insert(a_1, x)



2) House numbers



3) Basic program

```
10 print "hello"
15
20 print "how are you?"
30 GOTO 10
```

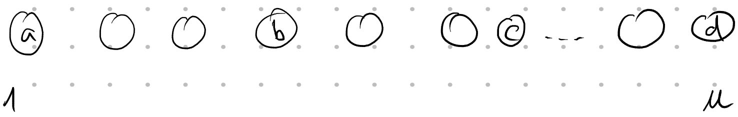
→ dynamic order maintenance
→ list labeling

[Dietz, Sleator 1987]

[Bender, Cole, Demaine, Farach-Colton, Zito, 2002]

n items, allocate $u = n^c$ slots

try to spread items "as evenly as possible"



$a \to b \to c \to d$

doubling/halving strategy

at beginning of each phase, set $u = n^c$, $N = n$, relabel everything as evenly as possible

- if n increases to $2N$ (after insert)
- if n decreases to $N/2$ (after delete)

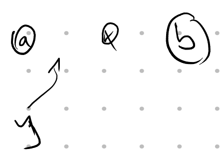
} start a new phase (relabel, etc.)

The actual cost of relabeling is $O(n)$, spread (amortised) across $\geq n/2$ operations within the last phase.

$\Rightarrow$ amortized cost $O(1)$

Summary: with $O(1)$ overhead we can assume that "globally" there is enough space.

... but "locally" we can still run out of space

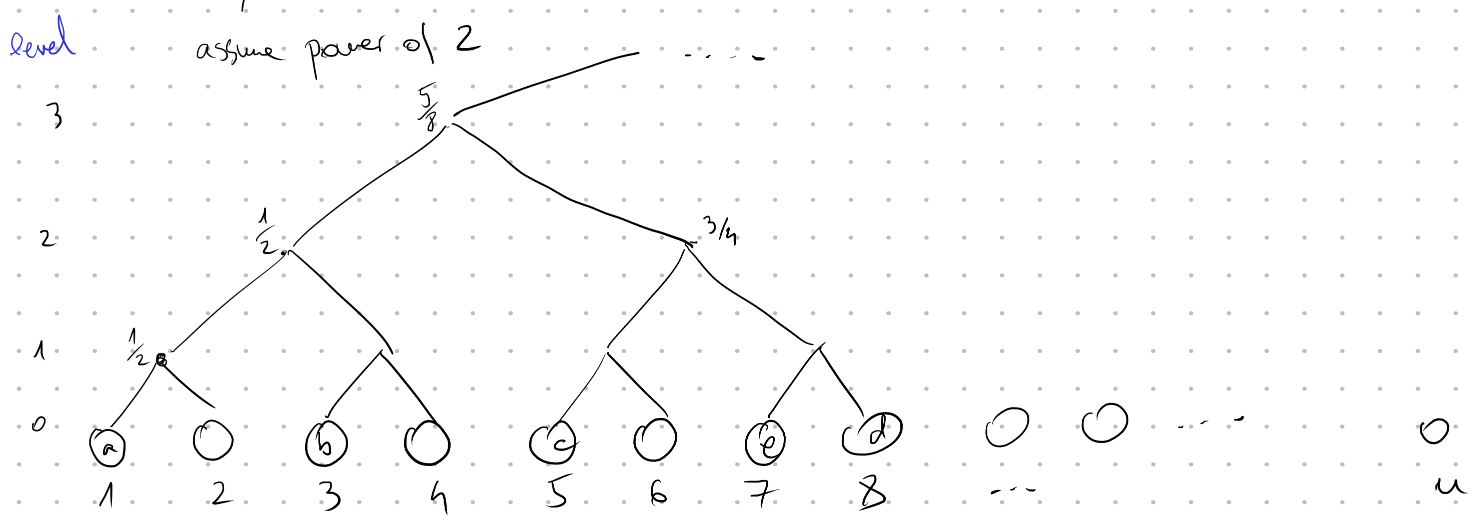


insert (a, x)

insert (a, y)

Think of $u = n^c$ slots as leaves of a complete BST (just conceptually)

assume power of 2



each internal node of level i has 2^i leaves in subtree

node density $= \dfrac{\#\text{ stored items in subtree}}{2^i}$

each internal node of level i has an "overflow density" $\underline{T_i}$

threshold where subtree is "too dense"

eg $\overset{1}{a} \to \overset{3}{b} \to \overset{5}{c} \to \overset{7}{e} \to \overset{8}{d}$

Operations:

order(x,y) $\longrightarrow$ compare $\ell(x)$ vs $\ell(y)$ $\quad O(1)$

delete(x) $\longrightarrow$ remove x from list $\quad O(1)$ amortized

if n "too small", start new phase, relabel everything

insert(x,y) link y after x, if n "too large", start new phase, relabel everything, etc

$$\cdots \to \overset{\ell(x)}{x} \to \overset{\ell(z)}{z} \to \cdots$$

if $\ell(z) > \ell(x)+1$

$$\cdots \to x \to y \to z \to \cdots$$

then assign y an arbitrary label

$\ell(x) < \ell(y) < \ell(z)$

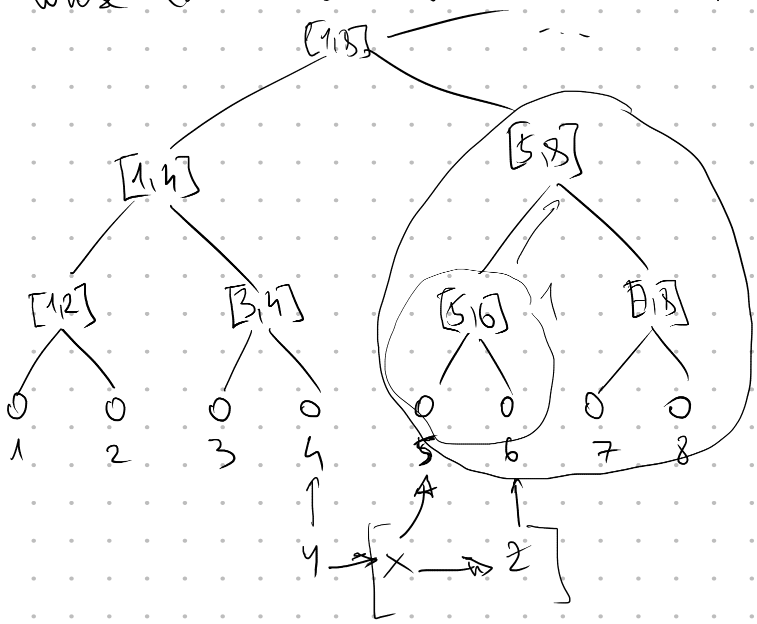
if $\ell(z) = \ell(x)+1$

then "walk up the tree from x" and find first node with density below its overflow density. $\to t$

Relabel all leaves in subtree of t, spacing labels evenly.

Set $\ell(y)$ to something between $\ell(x), \ell(z)$

<u>Note</u>: "Walking up tree" means examining items left/right of x in list, whose labels are within some range



overflow density threshold

$\frac{1}{1.5^3}$

$\frac{1}{1.5^2}$

$\frac{1}{1.5}$

1

$\ell = \ell(x)$

ith ancestor has range $\left[\left\lceil \frac{\ell}{2^i} \right\rceil \cdot 2^i - 2^i + 1, \left\lceil \frac{\ell}{2^i} \right\rceil \cdot 2^i\right]$

$5 \to [5,6] \to [5,8]$

$6 \to [5,6] \to [5,8]$

Remains to decide on overflow density T_i

$$T_0 = 1 = T^0 \qquad (\text{let } T \in (1,2), \text{ say } T = 1.5)$$

$$\dots$$

$$T_i = T^{-i} = \frac{1}{T^i}$$

The root must not overflow

$$\Rightarrow T_{\log_2 u} = T^{-\log_2 u} = u^{-\log_2 T} \overset{!!}{\geq} \frac{2n}{u}$$

(root will not overflow as long as $< 2n$ items are stored)

$\Rightarrow u^{1-\log_2 T} \geq 2n \quad | \log$

$$\log_2(u) \geq \left(\frac{1}{1-\log_2 T}\right)\log_2(2n)$$

$$\log_2(u) \geq \underbrace{\left(\frac{1}{1-\log_2 T}\right)}_{\in O(1)} \left(\log_2 n + 1\right)$$

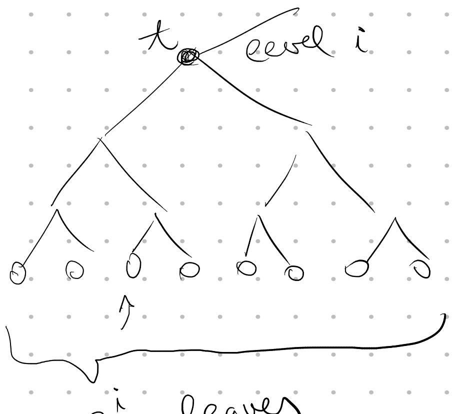
(T should be as small as possible)

sufficient to set $u \in n^{O(1)}$

order $O(1)$

delete $O(1)$

<u>Claim:</u> insert(x,y) takes $O(\log n)$ amortized time.



density of t is below overflow threshold

items stored in subtree

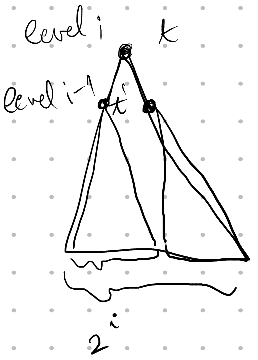
$$\frac{\#\text{items}}{2^i} < T_i = T^{-i}$$

Relabeling: assign labels s.t. gaps are

$$\left\lfloor \frac{2^i}{\#\text{items}} \right\rfloor$$

<u>Actual cost of relabeling</u>

items in subtree $< \frac{2^i}{T^i} = \left(\frac{2}{T}\right)^i$



Next relabel at t can happen only when a child of t will reach overflow density T_{i-1}.

At that time child of t (say t') will have in its subtree $\geq 2^{i-1} \cdot T_{i-1}$ items.

Currently, after relabeling t' has $\left(\frac{2}{T}\right)^i / 2$ items in its subtree (half of those in subtree of t)

So until next relabel at t there will be at least

$$2^{i-1} \cdot T_{i-1} - \frac{2^{i-1}}{T^i} \quad \text{inserts in subtree of } t'$$

$$= \frac{2^{i-1}}{T^{i-1}} - \frac{2^{i-1}}{T^i} = \frac{T\,2^{i-1} - 2^{i-1}}{T^i}$$

$$= \left(\frac{2}{T}\right)^i \cdot \frac{T-1}{2}$$

Actual cost of relabeling subtree of t (level i) is $\left(\frac{2}{T}\right)^i$

Spread across $\left(\frac{2}{T}\right)^i \cdot \frac{T-1}{2}$ operations (inserts below t)

Each such insert can deposit $\frac{2}{T-1} \in O(1)$ (set T as large as possible)

But each insert needs to deposit for all ancestors ($\log_2 n$ of them).

$\Rightarrow$ amortized cost of insert increases by

$$\log_2 n \cdot \frac{2}{T-1} \in O(\log n)$$