Dynamic order maintenance

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

operations:
- `insert(x, y)` → insert item y immediately after x
- `delete(x)` → delete item x
- `order(x, y)` → return true if x is before y, false otherwise

Example:
- `insert(a, b)` → a, b
- `insert(b, c)` → a, b, c
- `insert(a, d)` → a, d, b, c
- `order(a, c)` → true
- `order(b, a)` → false

idea 1: linked list

\[ a \rightarrow d \rightarrow b \rightarrow c \]

- `insert()` O(1)
- `delete()` O(1)
- `order()` O(n)

idea 2: balanced BST

- `insert(x, y)` O(log n) \(\geq O(n)\)
- `delete()` O(log n) \(\geq O(n)\)
- `order()` O(log n) in augmented trees

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

- `insert(a, d)` → a \(\rightarrow\) d \(\rightarrow\) b \(\rightarrow\) c
- `insert(a, e)` → a \(\rightarrow\) e \(\rightarrow\) d \(\rightarrow\) b \(\rightarrow\) c

\[ \text{eq.} \quad a \rightarrow b \rightarrow c \]

`order(x, y)`: compare labels
Problem: need too many bits for labels.

- New strategy: use integer labels, leave some gaps between labels.
  - Want labels to be \( \leq n^2 \), to only \( O(\log n) \) bits needed.

- Example:
  
  \[
  \begin{align*}
  10 & \rightarrow 20 \rightarrow 30 \\
  a & \rightarrow b \rightarrow c
  \end{align*}
  \]
  
  \( \text{rank}(a, b) \)
  
  \[
  \begin{align*}
  10 & \rightarrow 15 \rightarrow 20 \rightarrow 30 \\
  a & \rightarrow d \rightarrow b \rightarrow c
  \end{align*}
  \]

  If no place to insert new item,
  
  then we “loosen up” last by relabeling some items.

Goal:
- \( O(1) \) time for order \((x,y)\)
- \( O(1/m) \) amortized time for updates

(Also known as: “last label gap problem”)

Applications:

1. Items stored in memory

\( \text{rank}(a_1, x) \)

2. House numbers

\[
\begin{array}{c}
\text{1} & \text{2} & \text{3} & \text{4} & \text{5} \\
\text{a} & \text{b} & \text{c} & \text{d} & \text{...} \\
\end{array}
\]

3. Basic program

- 10  Print "Hello"
- 15  \hspace{1em} \text{dynamic order maintenance}
- 20  Print "How are you?"
- 30  GOTO 10

\( \Rightarrow \text{last label gap} \)
n items, allocate $U = n^e$ slots.

Try to spread items "as evenly as possible."

```
@ O O O b O O @ -- O @
```

**Doubling/halving strategy**

- $a \rightarrow b \rightarrow c \rightarrow a$

At beginning of each phase, set $U = n^e$, $N = n$, relabel everything as evenly as possible.

- if $n$ increases to $2N$ (after insert)
  - start a new phase
- if $n$ decreases to $N/2$ (after delete)
  - relabel, etc.

The actual cost of relabeling is $O(n)$, spread (amortised) across $\approx n^2$ 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.

```
@ O O b
\uparrow
@ \text{ammt}(a, x)
\uparrow
@ \text{ammt}(a, y)
```
Think of $n \approx n^2$ slots as leaves of a complete BST (just conceptually).

Level 3

- Assume powers of 2

- Each internal node of level $i$ has $2^i$ leaves in subtree

- Node density = \# stored items in subtree / $2^i$

- Each internal node of level $i$ has an "overflow density" $T_i$

- Threshold where subtree is "too dense"

- Operations:
  - \text{order}(x, y) \rightarrow \text{compare } \ell(x) \text{ vs } \ell(y) \quad O(1)
  - \text{delete}(x) \rightarrow \text{remove } x \text{ from list} \quad O(1) \text{ amortized}
  - \text{insert}(x, y) \text{ lead } y \text{ after } x \text{ if } n \text{ "too small" start new phase} \text{ relabel everything}
  - \text{insert}(x, y) \text{ lead } y \text{ after } x \text{ if } n \text{ "too large" start new phase} \text{ relabel everything}

- If $\ell(x) > \ell(y) + 1$

  - Then assign $y$ an arbitrary lead
    \[ \ell(x) < \ell(y) < \ell(z) \]

- If $\ell(z) = \ell(x) + 1$

  - Then "walk up the tree from $x$" and find first node with density below its overflow density.
Rounded all leaves in subtree of \( y \); specify labels evenly.

Set \( l(y) \) to something between \( l(x) \), \( l(z) \).

**Note:** "Walking up tree" means examining items left/right of \( x \) in list.

whose label are within some range.

\[
\left[ \frac{\ell}{2^i} - \frac{\ell}{2^{i+1}} \right]
\]

Let another box range

\[
5 \rightarrow [5, 6] \rightarrow [5, 8]
\]

\[
6 \rightarrow [5, 6] \rightarrow [5, 8]
\]

Reamins to decide an overflow density \( T_i \)

\[
T_0 = 1 = T^0 \quad \text{(let } T \in (1/2, 1) \text{ so } T = 1.5)\]

\[
T_i = T^{-i} = \frac{1}{T^i}
\]

The root must not overflow

\[
T_{\log_2 n} = \frac{1}{T^{\log_2 n}} \geq \frac{2n}{m} \quad \text{(root will not overflow as long as } m \geq 2n \text{ items are stored)}
\]
\[
\begin{align*}
\Rightarrow \quad & n - 1 \cdot \log_2 T \geq 2n / \log_2 T \\
\log_2(n) & \geq \left( \frac{1}{1 - \log_2 T} \right) \log_2 T \\
\log_2(n) & \geq \left( \frac{1}{1 - \log_2 T} \right) \log_2 n + 1 \\
& \leq O(1)
\end{align*}
\]
(\(T\) should be as small as possible)

It is sufficient to set \(n < \exp(O(1))\).

**Claim:** insert \((x,y)\) takes \(O(\log n)\) amortized time.

- **Array:** \(T\) is below overflow threshold
- **Items stored in subtree:**
  \[\frac{\text{# Items}}{2^i} \leq \frac{T_i}{T} > 1\]

- **Relabeling:** assign labels s.t. gaps are
  \[\left\lfloor \frac{2^i}{\# \text{items}} \right\rfloor\]

**Actual cost of relabeling**

\[\text{# items in subtree} < \frac{2^i}{T_i} = \left( \frac{2}{T} \right)^i\]
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 \( \left( \frac{2}{T} \right)^i \) items in its subtree
(half of those in subtree of \( t \)).

So until next relabel at \( t \) there will be at least
\[
2^{i-1} \cdot T_{i-1} - \frac{2^{i-1}}{T^2}
\]
inserts a subtree of \( t' \)
\[
= \frac{2^{i-1}}{T^{i-1}} - \frac{2^{i-1}}{T^i} = \frac{T \cdot 2^{i-1} - 2^{i-1}}{T^i}
\]
\[
= \left( \frac{2}{T} \right)^i \cdot \frac{T-1}{2}
\]
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 (insert 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 \( n \) of them).

\[ \text{= amortized cost of insert increases by} \]
\[ \log n \cdot \frac{2}{T-1} \in O(\log n) \]