Our goal: Best of hashing & trees

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Insert</th>
<th>Lookup</th>
<th>Successor</th>
<th>Ease of Prog.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted linked list</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>Easy</td>
</tr>
<tr>
<td>Unsorted linked list</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>Easy</td>
</tr>
<tr>
<td>Hash Table</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>Easy</td>
</tr>
<tr>
<td>Red/Black Tree</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>Hard</td>
</tr>
<tr>
<td>Binary Heap</td>
<td>O(1)</td>
<td>O(log n)</td>
<td>O(n)</td>
<td>Easy</td>
</tr>
<tr>
<td>Van Emde Boas Tree</td>
<td>O(lg lg n)</td>
<td>O(lg lg n)</td>
<td>O(lg lg n)</td>
<td>Harder (fun)</td>
</tr>
</tbody>
</table>

Key idea? D&C – But how? And where does O(lg lg n) come from?

The key idea: put a tree on top of array

Set \{1,9,10,15\}

How to split the tree: building intuition

- 100-story building
- 2 identical crystal balls
- Drop from diff. floors
  - Will not have a dent if k-1 floor drop
  - Will completely shatter if k floor drop
- How many throws to find k?
- One-ball strategy: 1,2,3,...,k \(\Rightarrow\) O(n)
- Two-ball strategy:
  - 1st ball big steps: \(h\); 2nd ball small steps: \(1\)
  - How to divide:
  - \(\min\{n/h+h\}\) when \(n/h=h\) and \(h\in\mathbb{N}\)
Putting it all together

Where might \(O(\sqrt{\lg u})\) bound arise?
- binary search over \(\sqrt{\lg u}\) elements
- recurrences: 
  \[ T'(\lg u) = T'(\frac{\lg u}{2}) + O(1) \]
  \[ T'(u) = T'(\frac{u}{2}) + O(1) \]

We'll develop van Emde Boas data structure by a series of improvements.

\[ T(u) = T(\frac{\sqrt{\lg u}}{2}) + O(1) \]
\[ T'(\lg u) = T'(\frac{\lg u}{2}) + O(1) \]
- let \(T'(\lg u) = T(v)\)
\[ T'(\lg u) = T'(\frac{1}{2} \lg u) + O(1) \]
- pull out the square root
\[ T'(x) = T'(\frac{1}{2} x) + O(1) \]
- substitute \(x = \lg u\)
\[ T' \] is \(O(\lg x)\) — using Master Method
\[ T' \] is \(O(\lg \lg u)\) — substitute for \(x = \lg u\)

Can we do better?

- Yes: instead of linear \(O(\sqrt{u})\) lookup in both the summary array and the sub-clusters, use the same data structure again recursively!
- Instead of only two levels only, recursive data structure with \(O(\lg \lg n)\) levels.

2) Split universe into clusters: \(\sqrt{u}\) of size \(\sqrt{u}\)

\[
\begin{array}{ccccccccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

- if \(x = i \cdot \sqrt{u} + j\) then \(V[x] = V.cluster[i][j]\)

\[
\text{define } \text{high}(x) = \lfloor x / \sqrt{u} \rfloor
\]

\[
\text{index}(i,j) = i \cdot \sqrt{u} + j
\]

= high & low-order halves in binary

3) Recurse: 3 ops in Successor are recursive Successors!

- \(V.cluster[i]\) is size-\(\sqrt{u}\) van Emde Boas \(0 \leq i < u\)
- \(V\) summary = size-\(\sqrt{u}\) van Emde Boas
- \(V\) summary[i] is \(V.cluster[i]\) nonempty?

\[ \text{Successor}(V, x): \]

\[ i = \text{high}(x) \]
\[ j = \text{Successor}(V.cluster[i], \text{low}(x)) \]

\[ \text{if } j = \infty:\]
\[ i = \text{Successor}(V\text{ summary, } i) \]
\[ j = \text{Successor}(V.cluster[i], -\infty) \]

\[ \text{return index}(i, j) \]

\[ \Rightarrow T(u) = 3T(\frac{u}{2}) + O(1) \]
\[ T'(\lg u) = 3T'(\frac{\lg u}{2}) + O(1) \]
\[ = O(\lg u^{3.322}) \]

- need to reduce to one recursion!
Summary: Van Emde Boas Trees

- Recursive data structure – Divide and Conquer
- Ideal splitting scenario with \( \sqrt{n} \) – crystal balls
- Limit number of recursive calls by min/max store
- Save space by only allocating non-empty clusters

- Only applicable when universe \( u \) is finite \( n^{\log k(n)} \)
- Powerful in theory, but cumbersome in practice
- Use for very large tables, then very efficient