Skip Lists in Julia

Drew Minnear

Massachusetts Institute of Technology

December 8, 2013
What are Skip Lists?

- Randomized data structure invented by William Pugh in the 80s.
What are Skip Lists?

- Randomized data structure invented by William Pugh in the 80s.
- Great for point and range queries in a set with an order.
What are Skip Lists?

- Randomized data structure invented by William Pugh in the 80s.
- Great for point and range queries in a set with an order.
- Insert, search, and delete all expected to be $O(\log n)!$
Inserting

- **Insert(skip, k)** inserts *k* into Skip List *skip*

  1. Search for the largest item on the bottom layer of *skip* that is less than or equal to *k*.
  2. Insert *k* on the bottom row after the element found in the search.
  3. Correct the linked list on this layer to include *k*.
  4. Flip a coin. If heads, move up a level and insert *k* above its location on the level below, making sure to preserve the correctness of the linked list.
  5. Repeat this step until tails is flipped.
Inserting

- \text{Insert}(\text{skip}, \ k) \text{ inserts } k \text{ into Skip List } \text{skip}
- \text{Search for largest item on bottom layer of skip that is less than or equal to } k.
Inserting

- **Insert**(skip, \( k \)) inserts \( k \) into Skip List \( \text{skip} \)
- Search for largest item on bottom layer of \( \text{skip} \) that is less than or equal to \( k \).
- Insert \( k \) on bottom row after element found in search. Correct linked list on this layer to include \( k \).
Inserting

- Insert(skip, $k$) inserts $k$ into Skip List skip
- Search for largest item on bottom layer of skip that is less than or equal to $k$.
- Insert $k$ on bottom row after element found in search. Correct linked list on this layer to include $k$.
- Flip a coin. If heads move up a level and insert $k$ above its location on the level below, making sure to preserve correctness of linked list. Repeat this step until a tails is flipped.
Searching

- Search(skip, \( k \)) returns true if \( k \) is in Skip List skip, false if it is not.
Searching

- Search(skip, k) returns true if k is in Skip List skip, false if it is not.
- Find largest item in skip that is less than or equal to k on top level. Drop down to analogous item on next lower level.
Searching

- Search(skip, k) returns true if k is in Skip List skip, false if it is not.
- Find largest item in skip that is less than or equal to k on top level. Drop down to analogous item on next lower level.
- Repeat the first step on the current level. Continuing repeating until k is found, in which case return true, or it is impossible to continue, in which case return false.
Deleting

- Delete(skip, k) deletes the first instance of \( k \) from skip.
Deleting

- **Delete**(*skip, k*) deletes the first instance of *k* from *skip*.
- **Search** for first instance of *k* in the bottom level of *skip*. 
Deleting

- Delete(skip, k) deletes the first instance of k from skip.
- Search for first instance of k in the bottom level of skip.
- Remove k from this level and repair linked list. Move up a level.
  Repeat this step until k is no longer in the current level.
Inserting

- Time to create data structure and insert 1,000,000 random integers in the range of Uint32.
Inserting

- Time to create data structure and insert 1,000,000 random integers in the range of Uint32.

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntSet</td>
<td>0.4679</td>
</tr>
<tr>
<td>Set</td>
<td>1.5373</td>
</tr>
<tr>
<td>Dict</td>
<td>0.2851</td>
</tr>
<tr>
<td>SkipList</td>
<td>35.9977</td>
</tr>
</tbody>
</table>
Searching for Item in Data Structure

- Time to conclude that an item is in data structure of 1,000,000 items. Does not include time to initialize data structure.
Searching for Item in Data Structure

- Time to conclude that an item is in data structure of 1,000,000 items. Does not include time to initialize data structure.

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntSet</td>
<td>$9.0 \times 10^{-6}$</td>
</tr>
<tr>
<td>Set</td>
<td>$1.28 \times 10^{-5}$</td>
</tr>
<tr>
<td>Dict</td>
<td>$9.0 \times 10^{-6}$</td>
</tr>
<tr>
<td>SkipList</td>
<td>$5.0 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
Searching for Item not in Data Structure

- Time to conclude that an item is not in data structure of 1,000,000 items. Does not include time to initialize data structure.
Searching for Item not in Data Structure

- Time to conclude that an item is not in data structure of 1,000,000 items. Does not include time to initialize data structure.

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntSet</td>
<td>$8.5 \times 10^{-6}$</td>
</tr>
<tr>
<td>Set</td>
<td>$9.8 \times 10^{-6}$</td>
</tr>
<tr>
<td>Dict</td>
<td>$9.0 \times 10^{-6}$</td>
</tr>
<tr>
<td>SkipList</td>
<td>$5.5 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
Deleting

- Time to remove an item from data structure of 1,000,000 items.
Deleting

- Time to remove an item from data structure of 1,000,000 items.

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntSet</td>
<td>$1.12 \times 10^{-5}$</td>
</tr>
<tr>
<td>Set</td>
<td>$1.60 \times 10^{-5}$</td>
</tr>
<tr>
<td>Dict</td>
<td>$1.22 \times 10^{-5}$</td>
</tr>
<tr>
<td>SkipList</td>
<td>$7.33 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
When Should Skip Lists be Used?

- For range queries. The other data structures are forced to search for each item in the range iteratively.
When Should Skip Lists be Used?

- For range queries. The other data structures are forced to search for each item in the range iteratively.
- Much faster to do this in a skip list. Consider if your range was real numbers between 1 and 10. (There are uncountably many.)
Distributed Skip Lists

- Distributed Skip Lists are a collection of Skip Lists on separate processes that act as a unified Skip List.
Distributed Skip Lists

- Distributed Skip Lists are a collection of Skip Lists on separate processes that act as a unified Skip List.
- Prior work has been done in the form of Skip Trees and Skip Tree Graphs.
Inserting

- Insert(dskip, k) inserts k into Distributed Skip List dskip
Inserting

- Insert(dskip, k) inserts k into Distributed Skip List dskip
- Randomly choose a process. Insert k into the skip list on that process.
Inserting

- \textbf{Insert}(dskip, \(k\)) inserts \(k\) into Distributed Skip List \(dskip\)
- Randomly choose a process. Insert \(k\) into the skip list on that process.
- \(O \left( \log \frac{n}{p} \right)\)
Searching

- Search(dskip, k) returns true if k is in Distributed Skip List dskip
Searching

- Search(dskip, k) returns true if k is in Distributed Skip List dskip
- Search for k in all processes. Reduce result with or.
Searching

- Search(dskip, k) returns true if k is in Distributed Skip List dskip
- Search for k in all processes. Reduce result with or.
- $O\left(\log\left(\frac{n}{p}\right) + p\right)$
Deleting

- **Delete(dskip, k)** removes an instance of *k* from Distributed Skip List *dskip*.
Deleting

- Delete(dskip, k) removes an instance of k from Distributed Skip List dskip.
- Search for k in all processes. Randomly pick a process to delete k from.
Deleting

- \text{Delete}(dskip, k)\text{ removes an instance of } k \text{ from Distributed Skip List } dskip.
- Search for \( k \) in all processes. Randomly pick a process to delete \( k \) from.
- \( O \left( \log \left( \frac{n}{p} \right) + p \right) \)