Typical LinkedList code

• Normally, have to have special case code for inserting or deleting at the front of a linked list, based on what needs to be updated:
  • the head of the list, or
  • the next pointer of the preceding element
General purpose data structure trick

- Use dummy node at the beginning of list
  - list head always points to same dummy node
  - doesn’t actually contain a value
- insertions/deletions *always* happen after some node
Step back for a moment

An alternative to binary trees trees
A data structure for sorted set

• Maintain an ordered set of elements
• Be able to find/insert/delete an element
• Iterate over the elements in order
• Find the next/previous element
• Can be a map too (keep a value associated with each key element)
Use a linked list, sorted by key
Operations

- Searching
  - scan list
    - if we find an element larger than the value we are looking for, we can abort

- Insert
  - scan list, looking for first element greater than or equal to element we are inserting
Insert

```java
boolean add(E e) {
    Node n = head;
    while (n.next != null && e.compareTo(n.next.key) > 0)
        n = n.next;
    if (n.next == null || e.compareTo(n.next.key) < 0) {
        n.next = new Node(e, n.next);
        return true;
    }
    return false; // e.compareTo(n.next.key) == 0
}
```
OK, it works, but is slow

- Searching element is slow (may take $O(n)$ time)
- If we are searching a dictionary for zebra, we don’t start at “A” and turn the pages one at a time
- Add an express lane
Use a linked list, sorted by key, with express lane
Search with an express lane

- Move in express lane until moving further in express lane would take us too far
- Move into local lane
- search as normal
boolean search(E e) {
    Node n = head;
    while (n.express != null
            && e.compareTo(n.express.key) > 0)
        n = n.express;
    while (n.local != null
            && e.compareTo(n.local.key) > 0)
        n = n.local;
    return n.local != null
          && e.compareTo(n.local.key) == 0;
}
Asymptotic complexity

• Unchanged
• We might only look at n/2 nodes
  • which is still $O(n)$
• But, it seems faster
• What if we add an express lane to the express lane
Use a linked list, sorted by key, with 2 express lanes.
Code

- Algorithms not much more complicated
- use next[lvl], with an outer loop counting lvl down
Search code

```java
boolean search(E e) {
    Node n = head;
    for(lvl = 2; lvl >= 0; lvl--)
        while (n.next[lvl] != null
            && e.compareTo(n.next[lvl].key) >= 0)
            n = n.next[lvl];
    return n != null
        && e.compareTo(n.key) == 0;
}
```
Asymptotic complexity

• Might only have to look at $n/4$ nodes
  • damn, still $O(n)$
• But still seems faster

• What if we keep adding levels, until the top level contains only a 1-2 nodes
Use a linked list, sorted by key, with many express lanes
Results?

- Code unchanged
  - except that we start at a different level
- Asymptotic of search is $O(\log n)$
  - essentially, a binary search
- But what about insert and delete?
Insert 49?
Insert 49?
Insert 49?

- Search still works just fine
- but we don’t maintain a consistent shape
- If we kept on inserting elements like this
- we won’t be $O(\log n)$
Notice something?
Notice something?
Notice something?

- \(\frac{1}{2}\) the elements have one link
Notice something?

- 1/2 the elements have one link
- 1/4 have two links
Notice something?

- 1/2 the elements have one link
- 1/4 have two links
- 1/8 have three links
Negative binomial distribution

- The levels have the same distribution as
- the number of times you have to flip a coin to get heads
- When you insert a number element, flip a coin until you get heads
  - the number of times you had to flip is the height of the inserted node
- slice the node into all appropriate lists
Randomized running time

• No matter what sequence of inputs you provide
• With very high probability, the running time is $O(\log n)$
• Based on random number generator, not on inputs
• You can prove this, and also bound the variance of the running time, and the probability curve of running time
Skip Lists

• This is skip lists, a data structure I invented in 1988

• A probabilistic alternative to balanced binary trees

• Better than (unbalanced) binary trees, which have horrible performance on some inputs

• Way easier to implement than balanced binary trees

• Have inspired lots of other data structures and algorithms
## 100,000 elements

### Random order

<table>
<thead>
<tr>
<th></th>
<th>searchTree</th>
<th>TreeMap</th>
<th>skip List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>insertion (ms)</strong></td>
<td>56</td>
<td>56</td>
<td>89</td>
</tr>
<tr>
<td><strong>search (ms)</strong></td>
<td>55</td>
<td>54</td>
<td>105</td>
</tr>
<tr>
<td><strong>lines of code</strong></td>
<td>100</td>
<td>330+</td>
<td>120</td>
</tr>
</tbody>
</table>
100,000 elements inserted in sorted order

<table>
<thead>
<tr>
<th></th>
<th>searchTree</th>
<th>TreeMap</th>
<th>skip List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>insertion</strong> (ms)</td>
<td>188,127</td>
<td>27</td>
<td>22</td>
</tr>
<tr>
<td><strong>search</strong> (ms)</td>
<td>114,937</td>
<td>68</td>
<td>100</td>
</tr>
</tbody>
</table>