Public Service Announcement

“The Experimental Social Science Laboratory (Xlab) invites you to participate in social science studies! Experiments conducted at Xlab (located in Hearst Gym, Suite 2) are computerized, decision-making studies such as tasks, surveys, and games. We also occasionally offer remote online and mobile studies that can be completed anywhere. Participants earn $15/hour on average every time they participate. For more information, visit xlab.berkeley.edu. To sign up, visit berkeley.sona-systems.com.”
CS61B Lecture #31

Today:

• More balanced search structures (DS(IJ), Chapter 9

Coming Up:

• Pseudo-random Numbers (DS(IJ), Chapter 11)
Really Efficient Use of Keys: the Trie

- Have been silent about cost of comparisons.
- For strings, worst case is length of string.
- Therefore should throw extra factor of key length, \( L \), into costs:
  - \( \Theta(M) \) comparisons really means \( \Theta(ML) \) operations.
  - So to look for key \( X \), keep looking at same chars of \( X \) \( M \) times.
- Can we do better? Can we get search cost to be \( O(L) \)?

Idea: Make a multi-way decision tree, with one decision per character of key.
The Trie: Example

- Set of keys
  \{a, abase, abash, abate, abbas, axolotl, axe, fabric, facet\}
- Ticked lines show paths followed for “abash” and “fabric”
- Each internal node corresponds to a possible prefix.
- Characters in path to node = that prefix.
Adding Item to a Trie

- Result of adding **bat** and **faceplate**.
- New edges ticked.
A Side-Trip: Scrunching

- For speed, obvious implementation for internal nodes is array indexed by character.
- \( O(L) \) performance, \( L \) length of search key.
- Looks as if independent of \( N \), number of keys. Is there a dependence?
- Problem: arrays are sparsely populated by non-null values—waste of space.

Idea: Put the arrays on top of each other!

- Use null (0, empty) entries of one array to hold non-null elements of another.
- Use extra markers to tell which entries belong to which array.
Scrunching Example

Small example:  (unrelated to Tries on preceding slides)

• Three leaf arrays, each indexed 0..9

A1:  
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>bass</td>
<td>trout</td>
<td>pike</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A2:  
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>ghee</td>
<td>milk</td>
<td>oil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A3:  
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>salt</td>
<td>cumin</td>
<td>mace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Now overlay them, but keep track of original index of each item:

A1:  
<table>
<thead>
<tr>
<th>0*</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5*</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>bass</td>
<td>trout</td>
<td>pike</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A2:  
<table>
<thead>
<tr>
<th>0</th>
<th>1*</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6*</th>
<th>7*</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>ghee</td>
<td>milk</td>
<td>oil</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A3:  
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9*</th>
</tr>
</thead>
<tbody>
<tr>
<td>salt</td>
<td>cumin</td>
<td>mace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A123:  
<table>
<thead>
<tr>
<th>0</th>
<th>-1</th>
<th>1</th>
<th>-1</th>
<th>2</th>
<th>5</th>
<th>5</th>
<th>7</th>
<th>6</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>bass</td>
<td>trout</td>
<td>pike</td>
<td>ghee</td>
<td>milk</td>
<td>oil</td>
<td>mace</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Last modified: Thu Nov 2 19:38:19 2017
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.

- More often thought of as an ordered list in which one can skip large segments.

- Typical example:

```
\begin{center}
\begin{tikzpicture}
  \node (0) at (0,0) {$\infty$};
  \node (1) at (1,0) {10};
  \node (2) at (2,0) {20};
  \node (3) at (3,0) {25};
  \node (4) at (4,0) {30};
  \node (5) at (5,0) {40};
  \node (6) at (6,0) {50};
  \node (7) at (7,0) {60};
  \node (8) at (8,0) {90};
  \node (9) at (9,0) {95};
  \node (10) at (10,0) {100};
  \node (11) at (11,0) {115};
  \node (12) at (12,0) {120};
  \node (13) at (13,0) {125};
  \node (14) at (14,0) {130};
  \node (15) at (15,0) {140};
  \node (16) at (16,0) {150};
  \node (17) at (17,0) {$\infty$};

  \draw[->] (0) -- (1);
  \draw[->] (1) -- (2);
  \draw[->] (2) -- (3);
  \draw[->] (3) -- (4);
  \draw[->] (4) -- (5);
  \draw[->] (5) -- (6);
  \draw[->] (6) -- (7);
  \draw[->] (7) -- (8);
  \draw[->] (8) -- (9);
  \draw[->] (9) -- (10);
  \draw[->] (10) -- (11);
  \draw[->] (11) -- (12);
  \draw[->] (12) -- (13);
  \draw[->] (13) -- (14);
  \draw[->] (14) -- (15);
  \draw[->] (15) -- (16);
  \draw[->] (16) -- (17);

  \node[draw, fill=black!20] at (1) {7};
  \node[draw, fill=black!20] at (2) {4};
  \node[draw, fill=black!20] at (3) {3};
  \node[draw, fill=black!20] at (4) {2};
  \node[draw, fill=black!20] at (5) {1};
  \node[draw, fill=black!20] at (6) {6};
  \node[draw, fill=black!20] at (7) {5};
  \node[draw, fill=black!20] at (8) {9};
  \node[draw, fill=black!20] at (9) {8};
  \node[draw, fill=black!20] at (10) {10};
  \node[draw, fill=black!20] at (11) {11};
  \node[draw, fill=black!20] at (12) {12};
  \node[draw, fill=black!20] at (13) {13};
  \node[draw, fill=black!20] at (14) {14};
  \node[draw, fill=black!20] at (15) {15};

\end{tikzpicture}
\end{center}
```

- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.

- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.

- Heights of the nodes were chosen randomly so that there are about $1/2$ as many nodes that are $> k$ high as there are that are $k$ high.

- Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:

  ![Skip List Diagram]

- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are $> k$ high as there are that are $k$ high.
- Makes searches fast with high probability.
A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.

More often thought of as an ordered list in which one can skip large segments.

Typical example:

To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.

In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.

Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are \( k \) high as there are that are \( k \) high.

Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

• A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.

• More often thought of as an ordered list in which one can skip large segments.

• Typical example:

![Diagram of a skip list example]

- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.

- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.

- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are $> k$ high as there are that are $k$ high.

- Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:

  ![Skip List Diagram]

  - To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
  - In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
  - Heights of the nodes were chosen randomly so that there are about \( \frac{1}{2} \) as many nodes that are \( > k \) high as there are that are \( k \) high.
  - Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:

```
0 1 2 3 10 20 25 30 40 50 55 60 90 95 100 115 120 125 130 140 150
```

- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are $k$ high as there are that are $k$ high.
- Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:

```
-∞ 0 1 2 3 10 20 25 30 40 50 55 60 90 95 100 115 120 125 130 140 150 ∞
```

- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about \(1/2\) as many nodes that are \(\geq k\) high as there are that are \(k\) high.
- Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at "random" heights.

- More often thought of as an ordered list in which one can skip large segments.

- Typical example:

  ![Skip List Diagram]

  - To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
  
  - In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
  
  - Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are > k high as there are that are k high.
  
  - Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A *skip list* can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.

- More often thought of as an ordered list in which one can skip large segments.

- Typical example:

  ![Diagram of a skip list]

  - To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
  - In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
  - Heights of the nodes were chosen randomly so that there are about $1/2$ as many nodes that are $\geq k$ high as there are that are $k$ high.
  - Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A skip list can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:

```
\[ \begin{array}{c}
\infty & 10 & 20 & 25 & 30 & 40 & 50 & 55 & 60 & 90 & 95 & 100 & 115 & 120 & 125 & 130 & 140 & 150 & \infty \\
0 & 1 & 2 & 3 & & & & & & & & & & & & & & \\
\end{array} \]
```

- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are \( k \) high as there are that are \( > k \) high.
- Makes searches fast with high probability.
Probabilistic Balancing: Skip Lists

- A *skip list* can be thought of as a kind of n-ary search tree in which we choose to put the keys at “random” heights.

- More often thought of as an ordered list in which one can skip large segments.

- Typical example:

  ![Skip List Diagram]

  - To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.

  - In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.

  - Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are \( k \) high as there are that are \( k \) high.

  - Makes searches fast with high probability.
Example: Adding and deleting

• Starting from initial list:

-∞
0  1  2  3
10 20 25 30 40 50 60 90 95 100 115 120 125 130 140 150 ∞

• In any order, we add 126 and 127 (choosing random heights for them), and remove 20 and 40:

-∞
0  1  2  3
10 25 30 50 55 60 90 95 100 115 120 126 127 130 140 150 ∞

• Shaded nodes here have been modified.
Summary

- Balance in search trees allows us to realize $\Theta(\log N)$ performance.
- B-trees, red-black trees:
  - Give $\Theta(\log N)$ performance for searches, insertions, deletions.
  - B-trees good for external storage. Large nodes minimize # of I/O operations
- Tries:
  - Give $\Theta(B)$ performance for searches, insertions, and deletions, where $B$ is length of key being processed.
  - But hard to manage space efficiently.
- Interesting idea: scrunched arrays share space.
- Skip lists:
  - Give probable $\Theta(\log N)$ performance for searches, insertions, deletions
  - Easy to implement.
  - Presented for interesting ideas: probabilistic balance, randomized data structures.
Summary of Collection Abstractions

- **Multiset**
  - contains, iterator
- **List**
  - `get(n)`
- **Set**
- **Ordered Set**
  - `first`
- **Unordered Set**
- **Priority Queue**
- **Sorted Set**
  - `subset`
- **Map**
  - contains, iterator
    - `get`
- **Unordered Map**
- **Ordered Map**

*Blue:* Java has corresponding interface
*Green:* Java has no corresponding interface

Last modified: Thu Nov 2 19:38:19 2017
Data Structures that Implement Abstractions

**Multiset**
- **List**: arrays, linked lists, circular buffers
- **Set**
  - **OrderedSet**
    - *Priority Queue*: heaps
    - *Sorted Set*: binary search trees, red-black trees, B-trees, sorted arrays or linked lists
  - **Unordered Set**: hash table

**Map**
- **Unordered Map**: hash table
- **Ordered Map**: red-black trees, B-trees, sorted arrays or linked lists
Corresponding Classes in Java

**Multiset** (Collection)

- **List**: ArrayList, LinkedList, Stack, ArrayBlockingQueue, ArrayDeque
- **Set**
  - OrderedSet
    - Priority Queue: PriorityQueue
    - Sorted Set (SortedSet): TreeSet
  - Unordered Set: HashSet

**Map**

- Unordered Map: HashMap
- Ordered Map (SortedMap): TreeMap