Armadillo story
Compiles?
BLEU score?
Time?
Memory?
### Data Types

<table>
<thead>
<tr>
<th>Max trigram count</th>
<th>468,261*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max bigram count</td>
<td>7,109,704*</td>
</tr>
<tr>
<td>Max unigram count</td>
<td>19,880,264</td>
</tr>
</tbody>
</table>

N₁ᵣ counts?  <=495,172

<table>
<thead>
<tr>
<th>MAX_INT</th>
<th>2,147,483,647</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_SHORT</td>
<td>32,767</td>
</tr>
</tbody>
</table>

This is appending <S> </S>

Try using a short + a byte or 3 bytes for trigrams (max val 8mil)
Load Factor - Memory

Buckets per unique key

Load Factor
Load Factor - Speed

Figure 5.12  Number of probes plotted against load factor for linear probing (dashed) and random strategy (S is successful search, U is unsuccessful search, and I is insertion)

Image Credit: Chris Brown
Hash Functions

\[
\text{key} = \text{key} \ ^ \ (\text{key} \ >> \ 32)
\]

\[
\begin{align*}
01101011 \\
00000110 \\
01101101
\end{align*}
\]

Ideally # collisions / number accesses = load factor

Most are good for random keys

Our keys aren’t random
   Tend to fall within a strict range
   Smaller values are more common

Shift by prime numbers
Modulo large prime number  Not too big or small
Avoiding Loops

\[
P(w_3 | w_1, w_2) = \frac{\max(c(w_3 | w_1, w_2) - d, 0)}{c(w_3 | w_2)} + \alpha(w_1, w_2) P(w_3 | w_2) = \\
\frac{\max(c(w_3 | w_1, w_2) - d, 0)}{c(w_3 | w_2)} + \alpha(w_1, w_2) P(w_3 | w_2)
\]

\[
P(w_3 | w_2) = \frac{\max(N_{1x} | \bullet, w_2 | \bullet) - d, 0}{\sum_{c \in \mathcal{C}} N_{1x} | \bullet, c} + \alpha(w_2) P(w_3 | w_2) = \\
\frac{\max(N_{1x} | \bullet, w_2 | \bullet) - d, 0}{N_{1x} | \bullet, \bullet} + \alpha(w_2) P(w_3 | w_2)
\]

\[
P(w_3) = \frac{N_{1x} | \bullet, w_3 | \bullet}{\sum_{c \in \mathcal{C}} N_{1x} | \bullet, c} = \frac{N_{1x} | \bullet, w_3 | \bullet}{N_{1x} | \bullet, \bullet}
\]

\[
\alpha(w_1, w_2) = d \cdot \frac{N_{1x} | w_1, w_2 | \bullet}{c(w_1, w_2)}
\]

\[
\alpha(w_2) = d \cdot \frac{N_{1x} | w_2 | \bullet}{N_{1x} | \bullet, w_2 | \bullet}
\]

Image Credit: Maria Ryskina
Multi-Value Hash Tables

Keys take up most amount of space
Re-use them by encoding many values for each
Multi-Value Hash Tables

Keys take up most amount of space
Re-use them by encoding many values for each
Do this for bigrams (trigrams don’t have extra counts, unigrams are arrays)
Byte Arrays (tentative)

BitSet
Caching

Frequent back to back queries for same n-gram
Things your friend told you to do that don’t work

• Being “clever” with initial size for your data structures
• Not storing unigram counts
• Using TIntOpenHashMap<T>

You will run out of memory during sanity checks!
Questions?