Tasks

1. Estimate a model from text
2. Query probabilities
Stupid Backoff

1. Count $n$-grams offline

2. Compute pseudo-probabilities at runtime

[Brants et al, 2007]
Stupid Backoff

1. Count $n$-grams offline

   \[ \text{count}(w_1^n) \]

2. Compute pseudo-probabilities at runtime

   \[
   \text{stupid}(w_n \mid w_1^{n-1}) = \begin{cases} 
   \frac{\text{count}(w_1^n)}{\text{count}(w_1^{n-1})} & \text{if } \text{count}(w_1^n) > 0 \\
   0.4 \text{stupid}(w_n \mid w_2^{n-1}) & \text{if } \text{count}(w_1^n) = 0 
   \end{cases}
   \]

   Note: stupid does not sum to 1.

[Brants et al, 2007]
Counting \( n \)-grams

\(<s>\) Australia is one of the few

\[\begin{array}{|c|c|}
\hline
\text{5-gram} & \text{Count} \\
\hline
\text{<s> Australia is one of} & 1 \\
\text{Australia is one of the} & 1 \\
\text{is one of the few} & 1 \\
\hline
\end{array}\]

Hash table from \( n \)-gram to count.
stupid($w_n \mid w_1^{n-1}$) = \begin{cases} 
\frac{\text{count}(w_1^n)}{\text{count}(w_1^{n-1})} & \text{if count}(w_1^n) > 0 \\
0.4 \text{stupid}(w_n \mid w_2^{n-1}) & \text{if count}(w_1^n) = 0
\end{cases}

stupid(few \mid is one of the) = \begin{cases} 
\text{count(is one of the few)} = 5 & \checkmark \\
\text{count(is one of the)} = 12
\end{cases}
stupid($w_n | w_{n-1}^n$) = \[ \begin{cases} \frac{\text{count}(w_{1}^n)}{\text{count}(w_{1}^{n-1})} & \text{if } \text{count}(w_{1}^n) > 0 \\ 0.4 \cdot \text{stupid}(w_n | w_{2}^{n-1}) & \text{if } \text{count}(w_{1}^n) = 0 \end{cases} \]

**stupid(periwinkle | is one of the)**

\[
\begin{align*}
\text{count(is one of the periwinkle)} &= 0 \times \\
\text{count(one of the periwinkle)} &= 0 \times \\
\text{count(of the periwinkle)} &= 0 \times \\
\text{count(the periwinkle)} &= 3 \checkmark \\
\text{count(the)} &= 1000
\end{align*}
\]
What’s Left?

- Hash table uses too much RAM
- Non-“stupid” smoothing methods (e.g. Kneser-Ney)
Save Memory: Forget Keys

Giant hash table with \( n \)-grams as keys and counts as values.

Replace the \( n \)-grams with 64-bit hashes:
Store hash(is one of) instead of “is one of”. Ignore collisions.
Save Memory: Forget Keys

Giant hash table with $n$-grams as keys and counts as values.

Replace the $n$-grams with 64-bit hashes:
Store hash(is one of) instead of “is one of”.
Ignore collisions.

Birthday attack: $\sqrt{2^{64}} = 2^{32}$.
$\implies$ Low chance of collision until $\approx 4$ billion entries.
Default Hash Table

`boost::unordered_map` and `__gnu_cxx::hash_map`

Diagram:
- `n-grams`
- Bucket array
  - 0 1 2 3 4 5
Default Hash Table

boost::unordered_map and __gnu_cxx::hash_map

$n$-grams

Bucket array

Lookup requires two random memory accesses.
Linear Probing Hash Table

- 1.5 buckets/entry (so buckets = 6).
- Ideal bucket = hash mod buckets.
- Resolve *bucket* collisions using the next free bucket.

<table>
<thead>
<tr>
<th>Words</th>
<th>Ideal</th>
<th>Hash</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>iran is</td>
<td>0</td>
<td>0x959e48455f4a2e90</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0</td>
<td>0</td>
</tr>
<tr>
<td>is one</td>
<td>2</td>
<td>0x186a7caef34acf16</td>
<td>5</td>
</tr>
<tr>
<td>one of</td>
<td>2</td>
<td>0xac66610314db8dac</td>
<td>2</td>
</tr>
<tr>
<td>&lt;s&gt; iran</td>
<td>4</td>
<td>0xf0ae9c2442c6920e</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0</td>
<td>0</td>
</tr>
</tbody>
</table>
Minimal Perfect Hash Table

Maps every $n$-gram to a unique integer $[0, |n-grams|)$
→ Use these as array offsets.
Minimal Perfect Hash Table

Maps every $n$-gram to a unique integer $[0, |n - \text{grams}|)$ → Use these as array offsets.

Entries not in the model get assigned offsets
⇒ Store a fingerprint of each $n$-gram
Minimal Perfect Hash Table

Maps every \( n \)-gram to a unique integer \([0, \vert n-grams \vert)\)

\[ \rightarrow \text{Use these as array offsets.} \]

Low memory, but potential for false positives
Sort $n$-grams, perform binary search.

Binary search is $O(|n\text{-grams}| \log |n\text{-grams}|)$. 
Sorted Array

Sort \( n \)-grams, perform binary search.

Binary search is \( O(|n\text{-grams}| \log |n\text{-grams}|) \).

Interpolation search is \( O(|n\text{-grams}| \log \log |n\text{-grams}|) \)
Lookups/µs
Entries

- probing
- hash_set
- unordered_set
- interpolation
- binary_search
- set
Reverse $n$-grams, arrange in a trie.

```
<ss> ----> is ----> Australia
  "one" ----> is ----> Australia
  "are" ----> are
  "are"
  "of"
```

Australia ----> <ss>
Saving More RAM

- Quantization: store approximate values
- Collapse probability and backoff
Conclusion

Implementation involves sparse mapping
- Hash table
- Probing hash table
- Minimal perfect hash table
- Sorted array with binary or interpolation search