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

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

Note: stupid does not sum to 1.

[Brants et al, 2007]

Counting *n*-grams

<s> Australia is one of the few

↓

5-**gram**	**Count**
<s> Australia is one of	1
Australia is one of the	1
is one of the few	1

Hash table from *n*-gram to count.

Query

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

stupid(few | is one of the)

count(is one of the few) $= 5$ ✓

count(is one of the) $= 12$

Query

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

stupid(periwinkle | is one of the)

count(is one of the periwinkle) = 0 ✗
count(one of the periwinkle) = 0 ✗
count(of the periwinkle) = 0 ✗
count(the periwinkle) = 3 ✓

count(the) = 1000

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

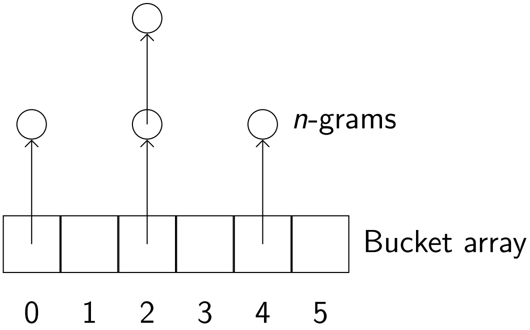
Giant hash table with *n*-grams as keys and counts as values.

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.

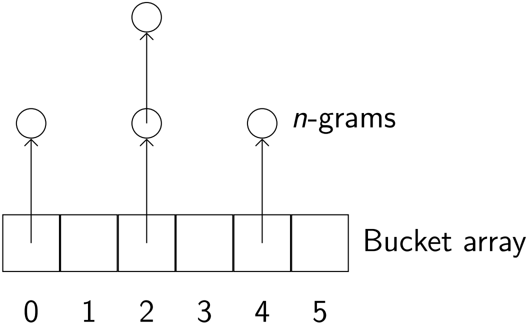
Default Hash Table

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



Default Hash Table

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



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.

Bigrams

Words	**Ideal**	**Hash**	**Count**
iran is	0	`0x959e48455f4a2e90`	3
		`0x0`	0
is one	2	`0x186a7caef34acf16`	5
one of	2	`0xac66610314db8dac`	2
<s> iran	4	`0xf0ae9c2442c6920e`	1
		`0x0`	0

Minimal Perfect Hash Table

Maps every *n*-gram to a unique integer $[0, |n - grams|)$
$\rightarrow$ Use these as array offsets.

Minimal Perfect Hash Table

Maps every *n*-gram to a unique integer $[0, |n - grams|)$
$\rightarrow$ Use these as array offsets.

Entries not in the model get assigned offsets
$\implies$ Store a fingerprint of each *n*-gram

Minimal Perfect Hash Table

Maps every *n*-gram to a unique integer $[0, |n - grams|)$
$\rightarrow$ Use these as array offsets.

Low memory, but potential for false positives

Sorted Array

Sort n-grams, perform binary search.

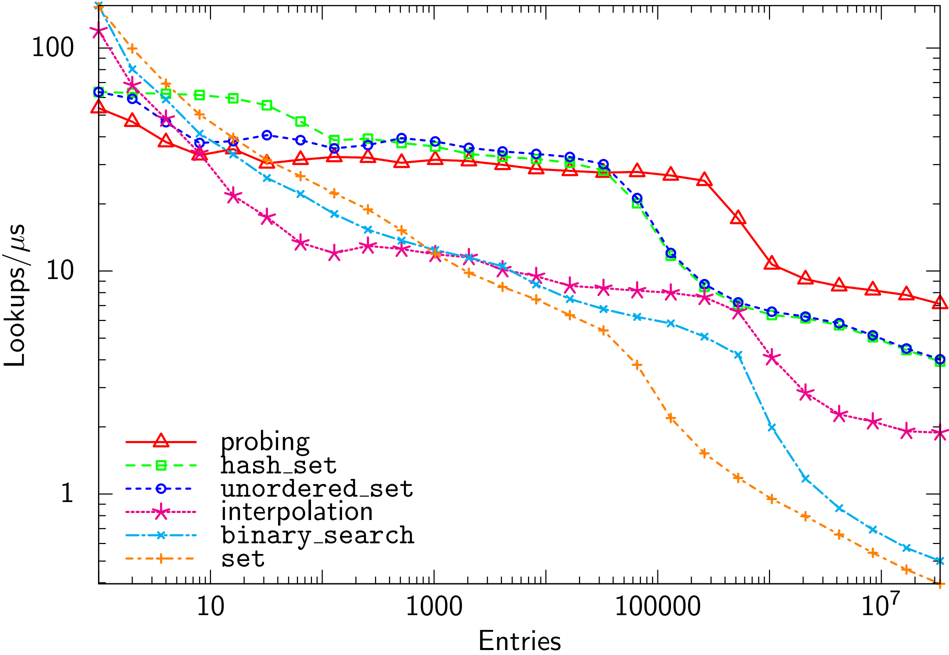
Binary search is $O(|n\text{-grams}| \log |n\text{-grams}|)$.

Sorted Array

Sort n-grams, perform binary search.

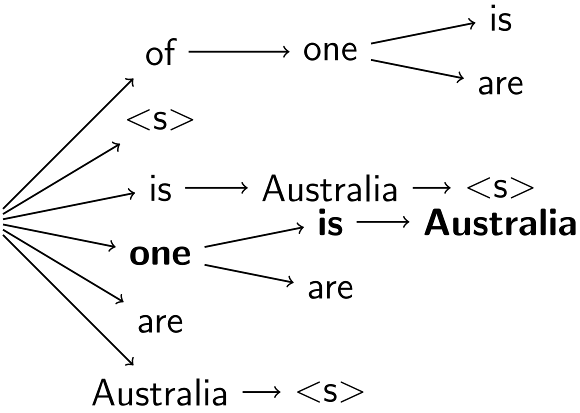
Binary search is $O(|n\text{-grams}| \log |n\text{-grams}|)$.

Interpolation search is $O(|n\text{-grams}| \log \log |n\text{-grams}|)$


Lookups/μs
100
10
1
10
1000
100000
10^7
Entries
probing
hash_set
unordered_set
interpolation
binary_search
set

Trie

Reverse *n*-grams, arrange in a trie.



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