Introduction to Information Retrieval
http://informationretrieval.org

IIR 3: Dictionaries and tolerant retrieval

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-10
Overview

1. Recap
2. Dictionaries
3. Wildcard queries
4. Edit distance
5. Spelling correction
6. Soundex
Outline

1. Recap
2. Dictionaries
3. Wildcard queries
4. Edit distance
5. Spelling correction
6. Soundex
Type/token distinction

- **Token** – an instance of a word or term occurring in a document
- **Type** – an equivalence class of tokens

*In June, the dog likes to chase the cat in the barn.*

- 12 word tokens, 9 word types
Problems in tokenization

- What are the delimiters? Space? Apostrophe? Hyphen?
- For each of these: sometimes they delimit, sometimes they don’t.
- No whitespace in many languages! (e.g., Chinese)
- No whitespace in Dutch, German, Swedish compounds (Lebensversicherungsgesellschaftsangestellter)
Problems with equivalence classing

- A term is an equivalence class of tokens.
- How do we define equivalence classes?
- Numbers (3/20/91 vs. 20/3/91)
- Case folding
- Stemming, Porter stemmer
- Morphological analysis: inflectional vs. derivational
- Equivalence classing problems in other languages
  - More complex morphology than in English
  - Finnish: a single verb may have 12,000 different forms
  - Accents, umlauts
Skip pointers

Brutus: 2 → 4 → 8 → 16 → 19 → 23 → 28 → 43

Caesar: 1 → 2 → 3 → 5 → 8 → 41 → 51 → 60 → 71
Positional indexes

- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and a list of positions
- Example query: “to₁ be₂ or₃ not₄ to₅ be₆”

**TO**, 993427:

\[
\langle 1: \langle 7, 18, 33, 72, 86, 231\rangle; \\
   2: \langle 1, 17, 74, 222, 255\rangle; \\
   4: \langle 8, 16, 190, 429, 433\rangle; \\
   5: \langle 363, 367\rangle; \\
   7: \langle 13, 23, 191\rangle; \ldots \rangle
\]

**BE**, 178239:

\[
\langle 1: \langle 17, 25\rangle; \\
   4: \langle 17, 191, 291, 430, 434\rangle; \\
   5: \langle 14, 19, 101\rangle; \ldots \rangle
\]

Document 4 is a match!
Positional indexes

- With a positional index, we can answer **phrase queries**.
- With a positional index, we can answer **proximity queries**.
Take-away
Tolerant retrieval: What to do if there is no exact match between query term and document term
Take-away

- **Tolerant retrieval**: What to do if there is no exact match between query term and document term
- Wildcard queries
Take-away

- **Tolerant retrieval**: What to do if there is no exact match between query term and document term
- Wildcard queries
- Spelling correction
Inverted index

For each term $t$, we store a list of all documents that contain $t$.

$\text{Brutus} \rightarrow 1 \ 2 \ 4 \ 11 \ 31 \ 45 \ 173 \ 174$

$\text{Caesar} \rightarrow 1 \ 2 \ 4 \ 5 \ 6 \ 16 \ 57 \ 132 \ \ldots$

$\text{Calpurnia} \rightarrow 2 \ 31 \ 54 \ 101$

\[\vdots\]

$\text{dictionary}$ \hspace{1cm} $\text{postings}$
Inverted index

For each term $t$, we store a list of all documents that contain $t$.

- **Brutus** → [1, 2, 4, 11, 31, 45, 173, 174]
- **Caesar** → [1, 2, 4, 5, 6, 16, 57, 132, ...]
- **Calpurnia** → [2, 31, 54, 101]

...
Dictionaries

- The dictionary is the data structure for storing the term vocabulary.
Dictionaries

- The dictionary is the data structure for storing the term vocabulary.
- **Term vocabulary:** the data
The dictionary is the data structure for storing the term vocabulary.

Term vocabulary: the data

Dictionary: the data structure for storing the term vocabulary
For each term, we need to store a couple of items:
For each term, we need to store a couple of items:
- document frequency
For each term, we need to store a couple of items:
  - document frequency
  - pointer to postings list
For each term, we need to store a couple of items:
- document frequency
- pointer to postings list
- ...
For each term, we need to store a couple of items:
- document frequency
- pointer to postings list
- ...

Assume for the time being that we can store this information in a fixed-length entry.
For each term, we need to store a couple of items:
- document frequency
- pointer to postings list

Assume for the time being that we can store this information in a fixed-length entry.

Assume that we store these entries in an array.
How do we look up a query term \( q_i \) in this array at query time? That is: which data structure do we use to locate the entry (row) in the array where \( q_i \) is stored?

<table>
<thead>
<tr>
<th>term</th>
<th>document frequency</th>
<th>pointer to postings list</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>656,265</td>
<td>→</td>
</tr>
<tr>
<td>aachen</td>
<td>65</td>
<td>→</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zulu</td>
<td>221</td>
<td>→</td>
</tr>
</tbody>
</table>

space needed: 20 bytes 4 bytes 4 bytes
Data structures for looking up term

- Two main classes of data structures: hashes and trees
Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  - Is there a fixed number of terms or will it keep growing?
Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  - Is there a fixed number of terms or will it keep growing?
  - What are the relative frequencies with which various keys will be accessed?
  - How many terms are we likely to have?
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array
- At query time: hash query term, locate entry in fixed-width array
Each vocabulary term is hashed into an integer, its row number in the array

At query time: hash query term, locate entry in fixed-width array

Pros: Lookup in a hash is faster than lookup in a tree.
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array.
- At query time: hash query term, locate entry in fixed-width array.
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array.
- At query time: hash query term, locate entry in fixed-width array.
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
- Cons
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array
- At query time: hash query term, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
- Cons
  - no way to find minor variants (resume vs. résumé)
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array
- At query time: hash query term, locate entry in fixed-width array
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
- Cons
  - no way to find minor variants (resume vs. résumé)
  - no prefix search (all terms starting with automat)
Hashes

- Each vocabulary term is hashed into an integer, its row number in the array.
- At query time: hash query term, locate entry in fixed-width array.
- Pros: Lookup in a hash is faster than lookup in a tree.
  - Lookup time is constant.
- Cons
  - No way to find minor variants (resume vs. résumé).
  - No prefix search (all terms starting with automat).
  - Need to rehash everything periodically if vocabulary keeps growing.
Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
Trees solve the prefix problem (find all terms starting with *automat*).

Simplest tree: binary tree
Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where $M$ is the size of the vocabulary.
Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where $M$ is the size of the vocabulary.
- $O(\log M)$ only holds for balanced trees.
Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where $M$ is the size of the vocabulary.
- $O(\log M)$ only holds for balanced trees.
- Rebalancing binary trees is expensive.
Trees solve the prefix problem (find all terms starting with *automat*).

- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where $M$ is the size of the vocabulary.
- $O(\log M)$ only holds for balanced trees.
- Rebalancing binary trees is expensive.
- **B-trees** mitigate the rebalancing problem.
Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- Search is slightly slower than in hashes: $O(\log M)$, where $M$ is the size of the vocabulary.
- $O(\log M)$ only holds for *balanced* trees.
- Rebalancing binary trees is expensive.
- **B-trees** mitigate the rebalancing problem.
- B-tree definition: every internal node has a number of children in the interval $[a, b]$ where $a, b$ are appropriate positive integers, e.g., $[2, 4]$. 
Binary tree

- Dictionaries
- Wildcard queries
- Edit distance
- Spelling correction
- Soundex

Schütze: Dictionaries and tolerant retrieval

19 / 114
B-tree
Outline

1 Recap
2 Dictionaries
3 Wildcard queries
4 Edit distance
5 Spelling correction
6 Soundex
Wildcard queries

- mon*: find all docs containing any term beginning with mon
Wildcard queries

- mon*: find all docs containing any term beginning with \textit{mon}
- Easy with B-tree dictionary: retrieve all terms $t$ in the range: $\text{mon} \leq t < \text{moo}$
Wildcard queries

- mon*: find all docs containing any term beginning with mon
- Easy with B-tree dictionary: retrieve all terms \( t \) in the range: \( \text{mon} \leq t < \text{moo} \)
- *mon: find all docs containing any term ending with mon
**Wildcard queries**

- mon*: find all docs containing any term beginning with *mon*
  - Easy with B-tree dictionary: retrieve all terms $t$ in the range: $\text{mon} \leq t < \text{moo}$
- *mon: find all docs containing any term ending with *mon*
  - Maintain an additional tree for terms *backwards*
Wildcard queries

- **mon*: find all docs containing any term beginning with *mon*
- Easy with B-tree dictionary: retrieve all terms $t$ in the range: $\text{mon} \leq t < \text{moo}$
- *mon: find all docs containing any term ending with *mon*
  - Maintain an additional tree for terms *backwards*
  - Then retrieve all terms $t$ in the range: $\text{nom} \leq t < \text{non}$
Wildcard queries

- **mon**: find all docs containing any term beginning with *mon*
  - Easy with B-tree dictionary: retrieve all terms $t$ in the range: $\text{mon} \leq t < \text{moo}$
- ***mon**: find all docs containing any term ending with *mon*
  - Maintain an additional tree for terms *backwards*
  - Then retrieve all terms $t$ in the range: $\text{nom} \leq t < \text{non}$
- Result: A set of terms that are matches for wildcard query
Wildcard queries

- $\text{mon}^*$: find all docs containing any term beginning with $\text{mon}$
  
  Easy with B-tree dictionary: retrieve all terms $t$ in the range: $\text{mon} \leq t < \text{moo}$

- $^*\text{mon}$: find all docs containing any term ending with $\text{mon}$
  
  Maintain an additional tree for terms $\text{backwards}$
  Then retrieve all terms $t$ in the range: $\text{nom} \leq t < \text{non}$

Result: A set of terms that are matches for wildcard query

Then retrieve documents that contain any of these terms
How to handle * in the middle of a term

Example: m*nchen
How to handle * in the middle of a term

- Example: m*nchen
- We could look up m* and *nchen in the B-tree and intersect the two term sets.
How to handle * in the middle of a term

- Example: m*ncchen
- We could look up m* and *ncchen in the B-tree and intersect the two term sets.
- Expensive
How to handle * in the middle of a term

- Example: m*nchen
- We could look up m* and *nchen in the B-tree and intersect the two term sets.
- Expensive
- Alternative: permuterm index
How to handle * in the middle of a term

- Example: m*nchen
- We could look up m* and *nchen in the B-tree and intersect the two term sets.
- Expensive
- Alternative: permuterm index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
How to handle * in the middle of a term

- Example: m*nchen
- We could look up m* and *nchen in the B-tree and intersect the two term sets.
- Expensive
- Alternative: permuterm index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
- Store each of these rotations in the dictionary, say, in a B-tree
For term **HELLO**: add *hello*$, *ello$h, *llo$he, *lo$hel, *o$hell*, and *$hello* to the B-tree where $ is a special symbol.
Permuterm $\rightarrow$ term mapping
Permuterm index

- For HELLO, we’ve stored: hello$, ello$h, llo$he, lo$hel, o$hell, $hello
Permuterm index

- For HELLO, we’ve stored: hello$, ello$h, llo$he, lo$hel, o$hell, $hello
- Queries
Permuterm index

- For HELLO, we’ve stored: hello$, ello$h, llo$he, lo$hel, o$hell, $hello

- Queries
  - For X, look up X$
Permuterm index

- For **HELLO**, we’ve stored: `hello$`, `ello$h`, `llo$he`, `lo$hel`, `o$hell`, `$hello`

- Queries
  - For X, look up X$
  - For X*, look up $X*$
Permuterm index

- For **HELLO**, we’ve stored: *hello$, ello$h, llo$he, lo$hel, o$hell, $hello*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X, look up X$$*
Permuterm index

- For **HELLO**, we’ve stored: *hello*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, *$hello*
- Queries
  - For *X*, look up *X*$
  - For *X*, look up *$X*"
  - For *X*, look up *X*$*
  - For *X*, look up *X*"
For HELLO, we’ve stored: hello$, ello$h, llo$he, lo$hel, o$hell, $hello

Queries
- For X, look up X$
- For X*, look up $X*
- For *X, look up X$*
- For *X*, look up X*
- For X*Y, look up Y$X*
Permuterm index

- For `HELLO`, we’ve stored: `hello$`, `ello$h`, `llo$he`, `lo$hel`, `o$hell`, `$hello`

- Queries
  - For `X`, look up `X$`
  - For `X*`, look up `$X*$`
  - For `*X`, look up `X$*`
  - For `*X*`, look up `X*`
  - For `X*Y`, look up `Y$X*`
  - Example: For `hel*o`, look up `o$hel*`
Permuterm index

- For **HELLO**, we’ve stored: *hello*, *ello$h*, *llo$he*, *lo$hel*, *o$hell*, *$hello*
- Queries
  - For X, look up X$
  - For X*, look up $X*
  - For *X*, look up X$
  - For *X*, look up X*
  - For X*Y, look up Y$X*
  - Example: For hel*o, look up o$hel*

Permuterm index would better be called a permuterm tree.
Permuterm index

- For **HELLO**, we’ve stored: \textit{hello$}, \textit{ello$h}, \textit{llo$he}, \textit{lo$hel}, \textit{o$hell}, \textit{$hello}

- Queries
  - For \textit{X}, look up \textit{X$}
  - For \textit{X*}, look up \textit{$X*}
  - For \textit{*X}, look up \textit{X$*}
  - For \textit{*X*}, look up \textit{X*}
  - For \textit{X*Y}, look up \textit{Y$X*}
  - Example: For \textit{hel*o}, look up \textit{o$hel*}

- Permuterm index would better be called a permuterm \textit{tree}.
- But permuterm index is the more common name.
Recap Dictionaries  Wildcard queries  Edit distance  Spelling correction  Soundex

Processing a lookup in the permuterm index

- Rotate query wildcard to the right
Processing a lookup in the permuterm index

- Rotate query wildcard to the right
- Use B-tree lookup as before
Processing a lookup in the permuterm index

- Rotate query wildcard to the right
- Use B-tree lookup as before
- Problem: Permuterm more than quadruples the size of the dictionary compared to a regular B-tree. (empirical number)
$k$-gram indexes

- More space-efficient than permuterm index
$k$-gram indexes

- More space-efficient than permuterm index
- Enumerate all character $k$-grams (sequence of $k$ characters) occurring in a term
**k-gram indexes**

- More space-efficient than permuterm index
- Enumerate all character $k$-grams (sequence of $k$ characters) occurring in a term
- 2-grams are called **bigrams**.
**$k$-gram indexes**

- More space-efficient than permuterm index
- Enumerate all character $k$-grams (sequence of $k$ characters) occurring in a term
- 2-grams are called **bigrams**.
- Example: from *April is the cruelest month* we get the bigrams:
  \[
  a\ ap\ pr\ ri\ il\ l\ i\ s\ s\ t\ th\ he\ e\ c\ cr\ ru\ ue\ el\ le\ es\ st\ t\ m\ mo\ on\ nt\ h\]

**k-gram indexes**

- More space-efficient than permuterm index
- Enumerate all character $k$-grams (sequence of $k$ characters) occurring in a term
- 2-grams are called **bigrams**.
- Example: from *April is the cruelest month* we get the bigrams:
  
  $a \ ap \ pr \ ri \ il \ l$ $i \ is \ s$ $t \ th \ he \ e$ $c \ cr \ ru \ ue \ el \ le \ es \ st \ t$ $m \ mo \ on \ nt \ h$

  - $\$$ is a special word boundary symbol, as before.
More space-efficient than permuterm index
Enumerate all character $k$-grams (sequence of $k$ characters) occurring in a term
2-grams are called **bigrams**.
Example: from *April is the cruelest month* we get the bigrams:
```
$a$ $a$ $a$ $a$ $p$ $p$ $p$ $p$ $r$ $r$ $r$ $r$ $i$ $i$ $i$ $i$ $i$ $i$ $l$ $l$ $l$ $l$ $s$ $s$ $s$ $s$ $s$ $s$ $s$ $t$ $t$ $t$ $t$ $t$ $t$ $t$ $t$ $t$ $m$
```

$\$ is a special word boundary symbol, as before.
Maintain an inverted index from bigrams to the terms that contain the bigram
Postings list in a 3-gram inverted index

etr → BEETROOT → METRIC → PETRIFY → RETRIEVAL
$k$-gram (bigram, trigram, ...) indexes

- Note that we now have two different types of inverted indexes
$k$-gram (bigram, trigram, \ldots) indexes

- Note that we now have two different types of inverted indexes
- The term-document inverted index for finding documents based on a query consisting of terms
Note that we now have two different types of inverted indexes:

- The term-document inverted index for finding documents based on a query consisting of terms.
- The $k$-gram index for finding terms based on a query consisting of $k$-grams.
Processing wildcarded terms in a bigram index

- Query mon* can now be run as:
  $m \text{ AND } mo \text{ AND on}$
Processing wildcarded terms in a bigram index

- Query mon* can now be run as:
  $m$ AND $mo$ AND $on$
- Gets us all terms with the prefix *mon* ...
Processing wildcarded terms in a bigram index

- Query mon* can now be run as:
  $m \text{ AND } mo \text{ AND on}$
- Gets us all terms with the prefix *mon* . . .
- . . . but also many “false positives” like *MOON*. 
Processing wildcarded terms in a bigram index

- Query `mon*` can now be run as: $m$ AND $mo$ AND on
- Gets us all terms with the prefix `mon` ...
- ...but also many “false positives” like `MOON`.
- We must postfilter these terms against query.
Processing wildcarded terms in a bigram index

- Query mon* can now be run as:
  $m$ AND $mo$ AND on
- Gets us all terms with the prefix mon . . .
- . . . but also many “false positives” like MOON.
- We must postfilter these terms against query.
- Surviving terms are then looked up in the term-document inverted index.
Query mon* can now be run as:
$m \text{ AND } mo \text{ AND } on$

- Gets us all terms with the prefix *mon*
- ... but also many "false positives" like *MOON*.
- We must postfilter these terms against query.
- Surviving terms are then looked up in the term-document inverted index.
- $k$-gram index vs. permuterm index
Query mon* can now be run as:
$\text{sm AND mo AND on}$

... but also many “false positives” like MOON.
We must postfilter these terms against query.
Surviving terms are then looked up in the term-document inverted index.

\textit{k-gram index vs. permuterm index}

\textit{k-gram index} is more space efficient.
Processing wildcarded terms in a bigram index

- Query `mon*` can now be run as:
  
  `$m \text{ AND } mo \text{ AND } on$`

- Gets us all terms with the prefix `mon` ...

- ... but also many “false positives” like `MOON`.

- We must postfilter these terms against query.

- Surviving terms are then looked up in the term-document inverted index.

- $k$-gram index vs. permuterm index
  
  - $k$-gram index is more space efficient.
  - Permuterm index doesn’t require postfiltering.
Exercise
Google has very limited support for wildcard queries.
Google has very limited support for wildcard queries. For example, this query doesn’t work very well on Google: 
[gen* universit*]
Exercise

- Google has very limited support for wildcard queries.
- For example, this query doesn’t work very well on Google: [gen* universit*]
  - Intention: you are looking for the University of Geneva, but don’t know which accents to use for the French words for university and Geneva.
Exercise

- Google has very limited support for wildcard queries.
- For example, this query doesn’t work very well on Google:
  \[\text{[gen* universit*]}\]
  - Intention: you are looking for the University of Geneva, but
    don’t know which accents to use for the French words for
    university and Geneva.
- According to Google search basics, 2010-04-29: “Note that
  the * operator works only on whole words, not parts of words.”
Exercise

- Google has very limited support for wildcard queries.
- For example, this query doesn’t work very well on Google: [gen* universit*]
  - Intention: you are looking for the University of Geneva, but don’t know which accents to use for the French words for university and Geneva.
- According to Google search basics, 2010-04-29: “Note that the * operator works only on whole words, not parts of words.”
- But this is not entirely true. Try [pythag*] and [m*nchen]
Exercise

- Google has very limited support for wildcard queries.
- For example, this query doesn’t work very well on Google: [gen* universit*]
  - Intention: you are looking for the University of Geneva, but don’t know which accents to use for the French words for university and Geneva.
- According to Google search basics, 2010-04-29: “Note that the * operator works only on whole words, not parts of words.”
- But this is not entirely true. Try [pythag*] and [m*nchen]
- Exercise: Why doesn’t Google fully support wildcard queries?
Processing wildcard queries in the term-document index
Problem 1: we must potentially execute a large number of Boolean queries.
Problem 1: we must potentially execute a large number of Boolean queries.

Most straightforward semantics: Conjunction of disjunctions
Processing wildcard queries in the term-document index

- Problem 1: we must potentially execute a large number of Boolean queries.
- Most straightforward semantics: Conjunction of disjunctions
- For [gen* universit*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR ...
Problem 1: we must potentially execute a large number of Boolean queries.

Most straightforward semantics: Conjunction of disjunctions

For \([\text{gen}^* \ \text{universit}^*]\): geneva university OR geneva université OR genève university OR genève université OR general universities OR . . .

Very expensive
Processing wildcard queries in the term-document index

- Problem 1: we must potentially execute a large number of Boolean queries.
- Most straightforward semantics: Conjunction of disjunctions
  - For [gen* universit*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR . . .
- Very expensive
- Problem 2: Users hate to type.
Problem 1: we must potentially execute a large number of Boolean queries.

Most straightforward semantics: Conjunction of disjunctions

For [gen* universit*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR ...

Very expensive

Problem 2: Users hate to type.

If abbreviated queries like [pyth* theo*] for [pythagoras’ theorem] are allowed, users will use them a lot.
Problem 1: we must potentially execute a large number of Boolean queries.

Most straightforward semantics: Conjunction of disjunctions

For \([\text{gen}* \text{universit}*]\): geneva university OR geneva université OR genève university OR genève université OR general universities OR...

Very expensive

Problem 2: Users hate to type.

If abbreviated queries like \([\text{pyth}^* \text{theo}^*]\) for [pythagoras’ theorem] are allowed, users will use them a lot.

This would significantly increase the cost of answering queries.
Problem 1: we must potentially execute a large number of Boolean queries.

Most straightforward semantics: Conjunction of disjunctions

For \([\text{gen}^* \text{ universit}^*]\): geneva university OR geneva université OR genève university OR genève université OR general universities OR . . .

Very expensive

Problem 2: Users hate to type.

If abbreviated queries like \([\text{pyth}^* \text{ theo}^*]\) for [pythagoras’ theorem] are allowed, users will use them a lot.

This would significantly increase the cost of answering queries.

Somewhat alleviated by Google Suggest
Outline

1 Recap
2 Dictionaries
3 Wildcard queries
4 Edit distance
5 Spelling correction
6 Soundex
Edit distance
The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$. 
The edit distance between string \( s_1 \) and string \( s_2 \) is the minimum number of basic operations that convert \( s_1 \) to \( s_2 \).

Levenshtein distance: The admissible basic operations are insert, delete, and replace.
The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$.

Levenshtein distance: The admissible basic operations are insert, delete, and replace

Levenshtein distance $\text{dog} - \text{do}$: 1
The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$.

Levenshtein distance: The admissible basic operations are insert, delete, and replace

Levenshtein distance $\text{dog-do}$: 1

Levenshtein distance $\text{cat-cart}$: 1
Edit distance

- The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$.
- Levenshtein distance: The admissible basic operations are insert, delete, and replace
  - Levenshtein distance $\text{dog-do}$: 1
  - Levenshtein distance $\text{cat-cart}$: 1
  - Levenshtein distance $\text{cat-cut}$: 1
Edit distance

- The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$.
- Levenshtein distance: The admissible basic operations are insert, delete, and replace
  - Levenshtein distance $\text{dog-do}$: 1
  - Levenshtein distance $\text{cat-cart}$: 1
  - Levenshtein distance $\text{cat-cut}$: 1
  - Levenshtein distance $\text{cat-act}$: 2
The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$.

- Levenshtein distance: The admissible basic operations are insert, delete, and replace.
- Levenshtein distance $dog$-$do$: 1
- Levenshtein distance $cat$-$cart$: 1
- Levenshtein distance $cat$-$cut$: 1
- Levenshtein distance $cat$-$act$: 2
- Damerau-Levenshtein distance $cat$-$act$: 1
Edit distance

- The edit distance between string $s_1$ and string $s_2$ is the minimum number of basic operations that convert $s_1$ to $s_2$.
- **Levenshtein distance**: The admissible basic operations are insert, delete, and replace
- Levenshtein distance $dog$-$do$: 1
- Levenshtein distance $cat$-$cart$: 1
- Levenshtein distance $cat$-$cut$: 1
- Levenshtein distance $cat$-$act$: 2
- **Damerau-Levenshtein distance** $cat$-$act$: 1
- Damerau-Levenshtein includes transposition as a fourth possible operation.
Levenshtein distance: Computation

<table>
<thead>
<tr>
<th></th>
<th>f</th>
<th>a</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Levenshtein distance: Algorithm

\[ \text{LEVENSHTEINDISTANCE}(s_1, s_2) \]

\begin{align*}
1 & \quad \textbf{for } i \leftarrow 0 \textbf{ to } |s_1| \\
2 & \quad \textbf{do } m[i, 0] = i \\
3 & \quad \textbf{for } j \leftarrow 0 \textbf{ to } |s_2| \\
4 & \quad \textbf{do } m[0, j] = j \\
5 & \quad \textbf{for } i \leftarrow 1 \textbf{ to } |s_1| \\
6 & \quad \textbf{do for } j \leftarrow 1 \textbf{ to } |s_2| \\
7 & \quad \quad \textbf{do if } s_1[i] = s_2[j] \\
8 & \quad \quad \quad \text{then } m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]\} \\
9 & \quad \quad \quad \text{else } m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1\} \\
10 & \quad \textbf{return } m[|s_1|, |s_2|] \\
\end{align*}

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)
Levenshtein distance: Algorithm

LevenshteinDistance($s_1$, $s_2$)

1. for $i \leftarrow 0$ to $|s_1|$
2. do $m[i, 0] = i$
3. for $j \leftarrow 0$ to $|s_2|$
4. do $m[0, j] = j$
5. for $i \leftarrow 1$ to $|s_1|$
6. do for $j \leftarrow 1$ to $|s_2|$
7. do if $s_1[i] = s_2[j]$
8. then $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]\}$
9. else $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1\}$
10. return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)
Levenshtein distance: Algorithm

**LevenshteinDistance**($s_1, s_2$)

1. **for** $i \leftarrow 0$ **to** $|s_1|$
2. **do** $m[i, 0] = i$
3. **for** $j \leftarrow 0$ **to** $|s_2|$
4. **do** $m[0, j] = j$
5. **for** $i \leftarrow 1$ **to** $|s_1|$
6. **do for** $j \leftarrow 1$ **to** $|s_2|$
7. **do if** $s_1[i] = s_2[j]$
8. **then** $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]\}$
9. **else** $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1\}$
10. **return** $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)
Levenshtein distance: Algorithm

\[
\text{LevenshteinDistance}(s_1, s_2)
\]

1. for \( i \leftarrow 0 \) to \( |s_1| \)
2. do \( m[i, 0] = i \)
3. for \( j \leftarrow 0 \) to \( |s_2| \)
4. do \( m[0, j] = j \)
5. for \( i \leftarrow 1 \) to \( |s_1| \)
6. do for \( j \leftarrow 1 \) to \( |s_2| \)
7. do if \( s_1[i] = s_2[j] \)
8. then \( m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]\} \)
9. else \( m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1\} \)
10. return \( m[|s_1|, |s_2|] \)

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)
Levenshtein distance: Algorithm

**LevenshteinDistance**($s_1$, $s_2$)

1. for $i \leftarrow 0$ to $|s_1|$ do $m[i, 0] = i$
2. for $j \leftarrow 0$ to $|s_2|$ do $m[0, j] = j$
3. for $i \leftarrow 1$ to $|s_1|$ do
   4. for $j \leftarrow 1$ to $|s_2|$ do
      5. if $s_1[i] = s_2[j]$ then $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]\}$
      6. else $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1\}$
6. return $m[|s_1|, |s_2|]$

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)
### Levenshtein distance: Example

<table>
<thead>
<tr>
<th></th>
<th>f</th>
<th>a</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>s</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Each cell of Levenshtein matrix

<table>
<thead>
<tr>
<th>cost of getting here from my upper left neighbor (copy or replace)</th>
<th>cost of getting here from my upper neighbor (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost of getting here from my left neighbor (insert)</td>
<td>the minimum of the three possible “movements”; the cheapest way of getting here</td>
</tr>
</tbody>
</table>
### Levenshtein distance: Example

<table>
<thead>
<tr>
<th></th>
<th>f</th>
<th>a</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>s</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Dynamic programming (Cormen et al.)
Dynamic programming (Cormen et al.)

- Optimal substructure: The optimal solution to the problem contains within it subsolutions, i.e., optimal solutions to subproblems.
Dynamic programming (Cormen et al.)

- Optimal substructure: The optimal solution to the problem contains within it subsolutions, i.e., optimal solutions to subproblems.
- Overlapping subsolutions: The subsolutions overlap. These subsolutions are computed over and over again when computing the global optimal solution in a brute-force algorithm.
Dynamic programming (Cormen et al.)

- Optimal substructure: The optimal solution to the problem contains within it sub-solutions, i.e., optimal solutions to sub-problems.
- Overlapping sub-solutions: The sub-solutions overlap. These sub-solutions are computed over and over again when computing the global optimal solution in a brute-force algorithm.
- Sub-problem in the case of edit distance: what is the edit distance of two prefixes
Dynamic programming (Cormen et al.)

- **Optimal substructure:** The optimal solution to the problem contains within it **subsolutions**, i.e., optimal solutions to subproblems.
- **Overlapping subsolutions:** The subsolutions overlap. These subsolutions are computed over and over again when computing the global optimal solution in a brute-force algorithm.
- **Subproblem in the case of edit distance:** what is the edit distance of two prefixes
- **Overlapping subsolutions:** We need most distances of prefixes 3 times – this corresponds to moving right, diagonally, down.
Weighted edit distance
Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., $m$ more likely to be mistyped as $n$ than as $q$. 
Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., $m$ more likely to be mistyped as $n$ than as $q$.
- Therefore, replacing $m$ by $n$ is a smaller edit distance than by $q$. 
Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., $m$ more likely to be mistyped as $n$ than as $q$.
- Therefore, replacing $m$ by $n$ is a smaller edit distance than by $q$.
- We now require a weight matrix as input.
Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., $m$ more likely to be mistyped as $n$ than as $q$.
- Therefore, replacing $m$ by $n$ is a smaller edit distance than by $q$.
- We now require a weight matrix as input.
- Modify dynamic programming to handle weights
Using edit distance for spelling correction
Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
- Intersect this set with our list of “correct” words
Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
- Intersect this set with our list of “correct” words
- Then suggest terms in the intersection to the user.
Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
- Intersect this set with our list of “correct” words
- Then suggest terms in the intersection to the user.
- → exercise in a few slides
Exercise

1. Compute Levenshtein distance matrix for OSLO – SNOW

2. What are the Levenshtein editing operations that transform cat into catcat?
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>l</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-----</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edit distance

Spelling correction

Soundex
<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td></td>
<td>1 1</td>
<td>2 2</td>
<td>3 3</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>1</td>
<td>1 2</td>
<td>2 3</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>2</td>
<td>2 1</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edit distance
<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Recap Dictionaries

#### Wildcard queries

#### Edit distance

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

#### Spelling correction

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

#### Soundex

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>-----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Recap
Dictionaries Wildcard queries

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1 1</td>
<td>2 2</td>
<td>3 3</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 2</td>
<td>2 3</td>
<td>2 4</td>
<td>4 5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 1</td>
<td>2 2</td>
<td>3 2</td>
<td>3 3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 2</td>
<td>2 3</td>
<td>3 3</td>
<td>3 4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3 1</td>
<td>2 2</td>
<td>3 3</td>
<td>4 3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 2</td>
<td>2 3</td>
<td>3 4</td>
<td>4 4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4 2</td>
<td>3 2</td>
<td>3 3</td>
<td>4 4</td>
<td></td>
</tr>
</tbody>
</table>
| 4 | 4 3 | 3 3 | 2 4 | 4 ？
| 4 | 5 3 | 4 3 | 4 | ？
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Edit distance**

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**Spelling correction**

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Soundex**

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
<td>w</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

This table illustrates the Edit distance for the words 'sno' and 'now', showing the minimum number of operations (insertions, deletions, or substitutions) required to transform one word into the other.
<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
How do I read out the editing operations that transform **OSLO** into **SNOW**?
<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**cost** | operation | **input** | **output**
---|---|---|---
0 | (copy) | o | o
1 | insert | * | w
### Edit distance

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td></td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>replace</td>
<td>l</td>
<td>n</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>s</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>n</th>
<th>o</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(copy)</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>1</td>
<td>replace</td>
<td>l</td>
<td>n</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>w</td>
</tr>
<tr>
<td>cost</td>
<td>operation</td>
<td>input</td>
<td>output</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>delete</td>
<td>o</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>1</td>
<td>replace</td>
<td>l</td>
<td>n</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>w</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>t</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

The table above illustrates the edit distance between different strings, where each cell represents the minimum number of insertions, deletions, or substitutions required to transform one string into another.
### Recap Dictionaries

#### Wildcard queries

- Edit distance
- Spelling correction
- Soundex

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>a</th>
<th>t</th>
<th>c</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

#### Edit distance

- Schütze: Dictionaries and tolerant retrieval

<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>t</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>
### Recap

<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(copy)</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>t</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>c</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

### Dictionaries

- Wildcard queries
- Edit distance
- Spelling correction
- Soundex

---

### Table

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>a</th>
<th>t</th>
<th>c</th>
<th>a</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

### Schütze: Dictionaries and tolerant retrieval

91 / 114
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(copy)</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>t</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>a</td>
<td>t</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>t</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cost</th>
<th>operation</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(copy)</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>(copy)</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>insert</td>
<td>*</td>
<td>t</td>
</tr>
</tbody>
</table>
Spelling correction
Spelling correction

- Two principal uses
Spelling correction

- Two principal uses
  - Correcting documents being indexed
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries
- Two different methods for spelling correction
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries
- Two different methods for spelling correction
  - **Isolated word** spelling correction
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries

- Two different methods for spelling correction
  - **Isolated word** spelling correction
    - Check each word on its own for misspelling
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries
- Two different methods for spelling correction
- **Isolated word** spelling correction
  - Check each word on its own for misspelling
  - Will not catch typos resulting in correctly spelled words, e.g.,
    
    \[ \text{an asteroid that fell form the sky} \]
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries

- Two different methods for spelling correction
  - **Isolated word** spelling correction
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words, e.g.,
      *an asteroid that fell form the sky*
  - **Context-sensitive** spelling correction
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries

- Two different methods for spelling correction
  - Isolated word spelling correction
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words, e.g.,
      *an asteroid that fell form the sky*
  - Context-sensitive spelling correction
    - Look at surrounding words
Spelling correction

- Two principal uses
  - Correcting documents being indexed
  - Correcting user queries

- Two different methods for spelling correction
  - **Isolated word** spelling correction
    - Check each word on its own for misspelling
    - Will not catch typos resulting in correctly spelled words, e.g.,
      *an asteroid that fell* *form* *the sky*
  
  - **Context-sensitive** spelling correction
    - Look at surrounding words
    - Can correct *form/from* error above
Correcting documents
Correcting documents

- We’re not interested in interactive spelling correction of documents (e.g., MS Word) in this class.
Correcting documents

- We’re not interested in interactive spelling correction of documents (e.g., MS Word) in this class.
- In IR, we use document correction primarily for OCR’ed documents. (OCR = optical character recognition)
Correcting documents

- We’re not interested in interactive spelling correction of documents (e.g., MS Word) in this class.
- In IR, we use document correction primarily for OCR’ed documents. (OCR = optical character recognition)
- The general philosophy in IR is: don’t change the documents.
Correcting queries
Correcting queries

- First: isolated word spelling correction
Correcting queries

- First: isolated word spelling correction
- Premise 1: There is a list of “correct words” from which the correct spellings come.
Correcting queries

- First: isolated word spelling correction
- Premise 1: There is a list of “correct words” from which the correct spellings come.
- Premise 2: We have a way of computing the distance between a misspelled word and a correct word.
Correcting queries

- First: isolated word spelling correction
- Premise 1: There is a list of “correct words” from which the correct spellings come.
- Premise 2: We have a way of computing the distance between a misspelled word and a correct word.
- Simple spelling correction algorithm: return the “correct” word that has the smallest distance to the misspelled word.
Correcting queries

- First: isolated word spelling correction
- Premise 1: There is a list of “correct words” from which the correct spellings come.
- Premise 2: We have a way of computing the distance between a misspelled word and a correct word.
- Simple spelling correction algorithm: return the “correct” word that has the smallest distance to the misspelled word.
- Example: *informaton* $\rightarrow$ *information*
Correcting queries

- First: isolated word spelling correction
- Premise 1: There is a list of “correct words” from which the correct spellings come.
- Premise 2: We have a way of computing the distance between a misspelled word and a correct word.
- Simple spelling correction algorithm: return the “correct” word that has the smallest distance to the misspelled word.
- Example: informaton → information
- For the list of correct words, we can use the vocabulary of all words that occur in our collection.
Correcting queries

- First: isolated word spelling correction

- Premise 1: There is a list of “correct words” from which the correct spellings come.

- Premise 2: We have a way of computing the distance between a misspelled word and a correct word.

- Simple spelling correction algorithm: return the “correct” word that has the smallest distance to the misspelled word.

- Example: informaton → information

- For the list of correct words, we can use the vocabulary of all words that occur in our collection.

- Why is this problematic?
Alternatives to using the term vocabulary
Alternatives to using the term vocabulary

- A standard dictionary (Webster’s, OED etc.)
Alternatives to using the term vocabulary

- A standard dictionary (Webster’s, OED etc.)
- An industry-specific dictionary (for specialized IR systems)
Alternatives to using the term vocabulary

- A standard dictionary (Webster’s, OED etc.)
- An industry-specific dictionary (for specialized IR systems)
- The term vocabulary of the collection, appropriately weighted
Distance between misspelled word and “correct” word
Distance between misspelled word and “correct” word

- Several alternatives
Distance between misspelled word and “correct” word

- Several alternatives
- Edit distance and Levenshtein distance
Distance between misspelled word and “correct” word

- Several alternatives
- Edit distance and Levenshtein distance
- Weighted edit distance
Distance between misspelled word and “correct” word

- Several alternatives
- Edit distance and Levenshtein distance
- Weighted edit distance
- $k$-gram overlap
Spelling correction
Now that we can compute edit distance: how to use it for isolated word spelling correction – this is the last slide in this section.
Now that we can compute edit distance: how to use it for isolated word spelling correction – this is the last slide in this section.

- $k$-gram indexes for isolated word spelling correction.
Now that we can compute edit distance: how to use it for isolated word spelling correction – this is the last slide in this section.

- $k$-gram indexes for isolated word spelling correction.
- Context-sensitive spelling correction
Now that we can compute edit distance: how to use it for isolated word spelling correction – this is the last slide in this section.

- $k$-gram indexes for isolated word spelling correction.
- Context-sensitive spelling correction
- General issues
$k$-gram indexes for spelling correction
$k$-gram indexes for spelling correction

- Enumerate all $k$-grams in the query term
**k-gram indexes for spelling correction**

- Enumerate all $k$-grams in the query term
- Example: bigram index, misspelled word *bordroom*
**k-gram indexes for spelling correction**

- Enumerate all $k$-grams in the query term
- Example: bigram index, misspelled word *bordroom*
- Bigrams: *bo, or, rd, dr, ro, oo, om*
$k$-gram indexes for spelling correction

- Enumerate all $k$-grams in the query term
- Example: bigram index, misspelled word *bordroom*
- Bigrams: *bo, or, rd, dr, ro, oo, om*
- Use the $k$-gram index to retrieve “correct” words that match query term $k$-grams
$k$-gram indexes for spelling correction

- Enumerate all $k$-grams in the query term
- Example: bigram index, misspelled word $bordroom$
- Bigrams: $bo$, $or$, $rd$, $dr$, $ro$, $oo$, $om$
- Use the $k$-gram index to retrieve “correct” words that match query term $k$-grams
- Threshold by number of matching $k$-grams
**$k$-gram indexes for spelling correction**

- Enumerate all $k$-grams in the query term
- Example: bigram index, misspelled word *bordroom*
- Bigrams: *bo, or, rd, dr, ro, oo, om*
- Use the $k$-gram index to retrieve “correct” words that match query term $k$-grams
- Threshold by number of matching $k$-grams
- E.g., only vocabulary terms that differ by at most 3 $k$-grams
$k$-gram indexes for spelling correction: bordroom
Context-sensitive spelling correction
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
  - Retrieve “correct” terms close to each query term
Our example was: *an asteroid that fell* **form** *the sky*

How can we correct **form** here?

One idea: **hit-based** spelling correction

- Retrieve “correct” terms close to each query term
- for *flew form munich*: **flea** for *flew*, **from** for **form**, **munch** for *munich*
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
  - Retrieve “correct” terms close to each query term
  - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
  - Now try all possible resulting phrases as queries with one word “fixed” at a time
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
  - Retrieve “correct” terms close to each query term
  - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
  - Now try all possible resulting phrases as queries with one word “fixed” at a time
  - Try query “*flea form munich*”
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
  - Retrieve “correct” terms close to each query term
  - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
  - Now try all possible resulting phrases as queries with one word “fixed” at a time
  - Try query “*flea form munich*”
  - Try query “*flew from munich*”
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
  - Retrieve “correct” terms close to each query term
  - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
  - Now try all possible resulting phrases as queries with one word “fixed” at a time
    - Try query “*flea form munich*”
    - Try query “*flew from munich*”
    - Try query “*flew form munch*”
Our example was: *an asteroid that fell* **form** *the sky*

How can we correct **form** here?

One idea: **hit-based** spelling correction

- Retrieve “correct” terms close to each query term
- For *flew form munich*: **flea** for *flew*, **from** for *form*, **munch** for *munich*

Now try all possible resulting phrases as queries with one word “fixed” at a time

- Try query “**flea form munich**”
- Try query “**flew from munich**”
- Try query “**flew form munch**”

The correct query “**flew from munich**” has the most hits.
Context-sensitive spelling correction

- Our example was: *an asteroid that fell form the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
  - Retrieve “correct” terms close to each query term
  - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
  - Now try all possible resulting phrases as queries with one word “fixed” at a time
    - Try query “*flea form munich*”
    - Try query “*flew from munich*”
    - Try query “*flew form munch*”
  - The correct query “*flew from munich*” has the most hits.
- Suppose we have 7 alternatives for *flew*, 20 for *form* and 3 for *munich*, how many “corrected” phrases will we enumerate?
Context-sensitive spelling correction
The “hit-based” algorithm we just outlined is not very efficient.
Context-sensitive spelling correction

- The “hit-based” algorithm we just outlined is not very efficient.
- More efficient alternative: look at “collection” of queries, not documents
General issues in spelling correction
General issues in spelling correction

- User interface
General issues in spelling correction

- User interface
  - automatic vs. suggested correction
General issues in spelling correction

- User interface
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
General issues in spelling correction

- **User interface**
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
General issues in spelling correction

- **User interface**
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
  - Tradeoff: simple vs. powerful UI
General issues in spelling correction

- **User interface**
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
  - Tradeoff: simple vs. powerful UI

- **Cost**
General issues in spelling correction

- User interface
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
  - Tradeoff: simple vs. powerful UI

- Cost
  - Spelling correction is potentially expensive.
General issues in spelling correction

- **User interface**
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
  - Tradeoff: simple vs. powerful UI

- **Cost**
  - Spelling correction is potentially expensive.
  - Avoid running on every query?
General issues in spelling correction

- **User interface**
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
  - Tradeoff: simple vs. powerful UI

- **Cost**
  - Spelling correction is potentially expensive.
  - Avoid running on every query?
  - Maybe just on queries that match few documents.
General issues in spelling correction

- **User interface**
  - automatic vs. suggested correction
  - *Did you mean* only works for one suggestion.
  - What about multiple possible corrections?
  - Tradeoff: simple vs. powerful UI

- **Cost**
  - Spelling correction is potentially expensive.
  - Avoid running on every query?
  - Maybe just on queries that match few documents.
  - Guess: Spelling correction of major search engines is efficient enough to be run on every query.
import re, collections

def words(text): return re.findall('^[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes = [a + b[1:] for a, b in splits if b]
    transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b) > 1]
    replaces = [a + c + b[1:] for a, b in splits for c in alphabet if b]
    inserts = [a + c + b for a, b in splits for c in alphabet]
    return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORDS)

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word) or [word]
    return max(candidates, key=NWORDS.get)
Outline

1 Recap

2 Dictionaries

3 Wildcard queries

4 Edit distance

5 Spelling correction

6 Soundex
Soundex

- Soundex is the basis for finding **phonetic** (as opposed to orthographic) alternatives.
Soundex

- Soundex is the basis for finding **phonetic** (as opposed to orthographic) alternatives.
- Example: *chebyshev / tchebyscheff*
Soundex is the basis for finding **phonetic** (as opposed to orthographic) alternatives.

Example: *chebyshev / tchebyscheff*

Algorithm:
Soundex is the basis for finding **phonetic** (as opposed to orthographic) alternatives.

**Example:** chebyshev / tchebyscheff

**Algorithm:**
- Turn every token to be indexed into a 4-character reduced form
Soundex

- Soundex is the basis for finding phonetic (as opposed to orthographic) alternatives.
- Example: `chebyshev / tchebyscheff`
- Algorithm:
  - Turn every token to be indexed into a 4-character reduced form
  - Do the same with query terms
Soundex

- Soundex is the basis for finding **phonetic** (as opposed to orthographic) alternatives.
- Example: *chebyshev / tchebyscheff*
- Algorithm:
  - Turn every token to be indexed into a 4-character reduced form
  - Do the same with query terms
  - Build and search an index on the reduced forms
Soundex algorithm

1. Retain the first letter of the term.
2. Change all occurrences of the following letters to '0' (zero): A, E, I, O, U, H, W, Y
3. Change letters to digits as follows:
   - B, F, P, V to 1
   - C, G, J, K, Q, S, X, Z to 2
   - D, T to 3
   - L to 4
   - M, N to 5
   - R to 6
4. Repeatedly remove one out of each pair of consecutive identical digits
5. Remove all zeros from the resulting string; pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits
Example: Soundex of HERMAN
Example: Soundex of *HERMAN*

- Retain H
Example: Soundex of *HERMAN*

- Retain H
- *ERMAN* → *ORM0N*
Example: Soundex of HERMAN

- Retain H
- \(ERMAN \rightarrow ORM0N\)
- \(ORM0N \rightarrow 06505\)
Example: Soundex of *HERMAN*

- Retain H
- $ERMAN \rightarrow ORM0N$
- $ORM0N \rightarrow 06505$
- $06505 \rightarrow 06505$
Example: Soundex of $HERMAN$

- Retain H
- $ERMAN \rightarrow ORM0N$
- $ORM0N \rightarrow 06505$
- $06505 \rightarrow 06505$
- $06505 \rightarrow 655$
Example: Soundex of *HERMAN*

- Retain H
- *ERMAN* → *ORM0N*
- *ORM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- Return *H655*
Example: Soundex of HERMAN

- Retain H
- ERMAN $\rightarrow$ ORM0N
- ORM0N $\rightarrow$ 06505
- 06505 $\rightarrow$ 06505
- 06505 $\rightarrow$ 655
- Return H655
- Note: HERMANN will generate the same code
How useful is Soundex?
How useful is Soundex?

- Not very – for information retrieval
How useful is Soundex?

- Not very – for information retrieval
- Ok for “high recall” tasks in other applications (e.g., Interpol)
How useful is Soundex?

- Not very – for information retrieval
- Ok for “high recall” tasks in other applications (e.g., Interpol)
- Zobel and Dart (1996) suggest better alternatives for phonetic matching in IR.
Exercise

- Compute Soundex code of your last name
Take-away

- **Tolerant retrieval**: What to do if there is no exact match between query term and document term
- Wildcard queries
- Spelling correction
Resources

- Chapter 3 of IIR
- Resources at http://cislmu.org
  - trie vs hash vs ternary tree
  - Soundex demo
  - Edit distance demo
  - Peter Norvig’s spelling corrector
  - Google: wild card search, spelling correction gone wrong, a misspelling that is more frequent that the correct spelling