

# Introduction to Information Retrieval

## IIR 3: Dictionaries and tolerant retrieval

Mihai Surdeanu

(Based on slides by Hinrich Schütze at [informationretrieval.org](http://informationretrieval.org))

Fall 2015

# 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
- **Token** – unique tokens that go in the dictionary
- **Type** – an equivalence class of tokens
- *In June, the dog likes to chase the cat in the barn.*
- 12 word tokens, 9 terms (or word types)

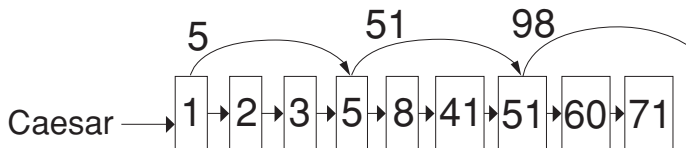
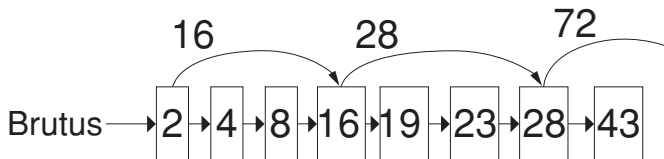
# Problems in tokenization

- 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



# 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: *"to<sub>1</sub> be<sub>2</sub> or<sub>3</sub> not<sub>4</sub> to<sub>5</sub> be<sub>6</sub>"*

TO, 993427:

< 1: <7, 18, 33, 72, 86, 231>;  
 2: <1, 17, 74, 222, 255>;  
 4: <8, 16, 190, 429, 433>;  
 5: <363, 367>;  
 7: <13, 23, 191>; ... >

BE, 178239:

< 1: <17, 25>;  
 4: <17, 191, 291, 430, 434>;  
 5: <14, 19, 101>; ... >

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
- Wildcard queries
- Spelling correction

# Outline

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

# 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
-----------	---	---	----	----	-----

⋮

**dictionary**

**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
-----------	---	---	----	----	-----

⋮

**dictionary**

**postings**

# Dictionaries

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

# Dictionary as array of fixed-width entries

- 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.

# Dictionary as array of fixed-width entries

term	document frequency	pointer to postings list
a	656,265	→
aachen	65	→
...	...	...
zulu	221	→

space needed: 20 bytes 4 bytes 4 bytes

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?



# Data structures for looking up term

- Two main classes of data structures: hashes and trees
- Some IR systems use hashes, some use trees.

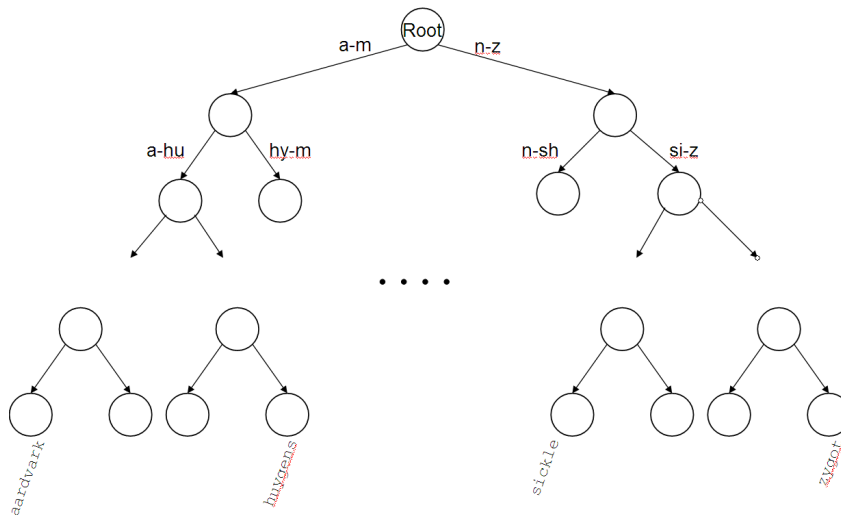
# Hashes

- 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

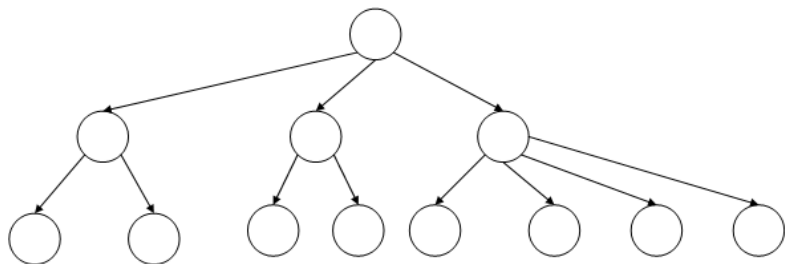
# Trees

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



# B-tree



# Outline

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

# Wildcard queries

- Trailing wildcard:  $mon^*$ : find all docs containing any term beginning with *mon*
  - Binary tree: traverse down the symbols:  $m > o > n$
  - Easy with B-tree dictionary: retrieve all terms  $t$  in the range:  
 $mon \leq t < moo$
- Leading wildcard:  $*mon$ : find all docs containing any term ending with *mon*
  - How would you do this?

# Wildcard queries

- Trailing wildcard:  $mon^*$ : find all docs containing any term beginning with *mon*
  - Binary tree: traverse down the symbols:  $m > o > n$
  - Easy with B-tree dictionary: retrieve all terms  $t$  in the range:  
 $mon \leq t < moo$
- Leading wildcard:  $^*mon$ : find all docs containing any term ending with *mon*
  - How would you do this?
  - Maintain an additional tree for terms *backwards*
  - Then retrieve all terms  $t$  in the range:  $nom \leq t < non$
- 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
- We could look up m\* and \*nchen in the B-tree and intersect the two term sets.
- Expensive

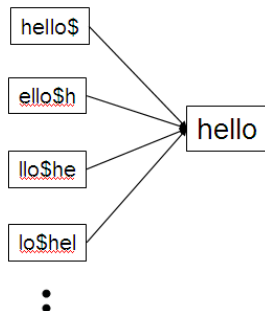
# How to handle \* in the middle of a term

- Example: m\*nchen
- We could look up m\* and \*nchen in the B-tree and intersect the two term sets.
- Expensive
- Alternative: [permuterm](#) index
- 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

# Permuterm index

- For term HELLO: add *hello\$*, *ello\$h*, *llo\$he*, *lo\$hel*, *o\$hell*, and *\$hello* to the B-tree where \$ is a special symbol
- This is done at indexing time for every word!

# Permuterm → term mapping



# 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\*Y, look up Y\$X\*
  - Example: For hel\*o, look up o\$hel\*
  - How about the query fi\*mo\*er?
- “Permuterm index” is a really bad name! It would better be called a permuterm [tree](#).
- But permuterm index is the more common name.

# 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
- 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. **But we use it now to mark both beginning and ending of words!**
- Maintain an inverted index from bigrams to the terms that contain the bigram

# Postings list in a 3-gram inverted index





# $k$ -gram (bigram, trigram, ...) 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
- 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 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.
- *k*-gram index vs. permuterm index
  - *k*-gram index is more space efficient.
  - Permuterm index doesn't require postfiltering.

# Exercise

- Google has 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.
  - These actually work well now!
- According to Google search basics, 2010-04-29: "Note that the \* operator works only on whole words, not parts of words."
- Exercise: Why doesn't (didn'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.
- 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.
- This would significantly increase the cost of answering queries.
- Somewhat alleviated by Google Suggest

## More Exercises

- Exercise 3.3
- Exercise 3.4
- Exercise 3.5

# Outline

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

# 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

- 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

- 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*
- 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

- 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

- We will study several alternatives.
- Edit distance and Levenshtein distance
- Weighted edit distance
- $k$ -gram overlap

# 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

		f	a	s	t
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	2	2
s	4	4	3	2	3

# 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

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  $m[i, j] = \min\{\textcolor{red}{m[i-1, j]}+1, m[i, j-1]+1, m[i-1, j-1]\}$ 
9          else  $m[i, j] = \min\{\textcolor{red}{m[i-1, j]}+1, m[i, j-1]+1, m[i-1, j-1]+1\}$ 
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

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: Example

			f		a		s		t	
		<u>0</u>	<u>1</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>
c		<u>1</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>5</u>
		<u>1</u>	<u>2</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>
a		<u>2</u>	<u>2</u>	<u>2</u>	<u>1</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>5</u>
		<u>2</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>
t		<u>3</u>	<u>3</u>	<u>3</u>	<u>3</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>2</u>	<u>4</u>
		<u>3</u>	<u>4</u>	<u>3</u>	<u>4</u>	<u>2</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>2</u>
s		<u>4</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>3</u>
		<u>4</u>	<u>5</u>	<u>4</u>	<u>5</u>	<u>3</u>	<u>4</u>	<u>2</u>	<u>3</u>	<u>3</u>

# Each cell of Levenshtein matrix

cost of getting here from my upper left neighbor (copy or replace)	cost of getting here from my upper neighbor (delete)
cost of getting here from my left neighbor (insert)	the minimum of the three possible “movements”; the cheapest way of getting here

# Levenshtein distance: Example

			f		a		s		t	
		<u>0</u>	<u>1</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>
c		<u>1</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>5</u>
		<u>1</u>	<u>2</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>
a		<u>2</u>	<u>2</u>	<u>2</u>	<u>1</u>	<u>3</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>5</u>
		<u>2</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>1</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>3</u>
t		<u>3</u>	<u>3</u>	<u>3</u>	<u>3</u>	<u>2</u>	<u>2</u>	<u>3</u>	<u>2</u>	<u>4</u>
		<u>3</u>	<u>4</u>	<u>3</u>	<u>4</u>	<u>2</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>2</u>
s		<u>4</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>3</u>	<u>2</u>	<u>3</u>	<u>3</u>	<u>3</u>
		<u>4</u>	<u>5</u>	<u>4</u>	<u>5</u>	<u>3</u>	<u>4</u>	<u>2</u>	<u>3</u>	<u>3</u>

# 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

- 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

- 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 the next slide



# Exercise

- 1 Compute Levenshtein distance matrix for OSLO – SNOW
- 2 What are the Levenshtein editing operations that transform *cat* into *catcat*?

		s	n	o	w
	<hr/> 0	<hr/> 1 1	<hr/> 2 2	<hr/> 3 3	<hr/> 4 4
o	<hr/> 1 1				
s	<hr/> 2 2				
l	<hr/> 3 3				
o	<hr/> 4 4				

		s	n	o	w
	<u>  </u> 0	<u>  1  </u> 1 1	<u>  2  </u> 2 2	<u>  3  </u> 3 3	<u>  4  </u> 4 4
o	<u>  1  </u> 1	<u>  1  </u> 2	<u>  2  </u> ?		
s	<u>  2  </u> 2				
l	<u>  3  </u> 3				
o	<u>  4  </u> 4				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>			
s	<u>2</u> <u>2</u>				
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<hr/> 0	<hr/> 1 1	<hr/> 2 2	<hr/> 3 3	<hr/> 4 4
o	<hr/> 1 1	<hr/> 1 2 2 1	<hr/> 2 3 2 ?		
s	<hr/> 2 2				
l	<hr/> 3 3				
o	<hr/> 4 4				

		s	n	o	w
	<u>  </u> 0	<u>  1  </u> 1	<u>  2  </u> 2	<u>  3  </u> 3	<u>  4  </u> 4
o	<u>  1  </u> 1	<u>  1  2  </u> 2 1	<u>  2  3  </u> 2 2		
s	<u>  2  </u> 2				
l	<u>  3  </u> 3				
o	<u>  4  </u> 4				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>?</u>	
s	<u>2</u> <u>2</u>				
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	
s	<u>2</u> <u>2</u>				
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				



		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>?</u>
s	<u>2</u> <u>2</u>				
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>				
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>	<u>1</u> <u>2</u> <u>3</u> <u>?</u>			
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>	<u>1</u> <u>2</u> <u>3</u> <u>1</u>			
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> 1	<u>1 2</u> 2 1	<u>2 3</u> 2 2	<u>2 4</u> 3 2	<u>4 5</u> 3 3
s	<u>2</u> 2	<u>1 2</u> 3 1	<u>2 3</u> 2 ?		
l	<u>3</u> 3				
o	<u>4</u> 4				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>	<u>1</u> <u>2</u> <u>3</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>		
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> 1	<u>1 2</u> 2 1	<u>2 3</u> 2 2	<u>2 4</u> 3 2	<u>4 5</u> 3 3
s	<u>2</u> 2	<u>1 2</u> 3 1	<u>2 3</u> 2 2	<u>3 3</u> 3 ?	
l	<u>3</u> 3				
o	<u>4</u> 4				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> 1	<u>1 2</u> 2 1	<u>2 3</u> 2 2	<u>2 4</u> 3 2	<u>4 5</u> 3 3
s	<u>2</u> 2	<u>1 2</u> 3 1	<u>2 3</u> 2 2	<u>3 3</u> 3 3	
l	<u>3</u> 3				
o	<u>4</u> 4				



		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 ?</u>
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>	<u>1</u> <u>2</u> <u>3</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>3</u> <u>3</u> <u>3</u> <u>3</u>	<u>3</u> <u>4</u> <u>4</u> <u>3</u>
l	<u>3</u> <u>3</u>				
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>	<u>1</u> <u>2</u> <u>3</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>3</u> <u>3</u> <u>3</u> <u>3</u>	<u>3</u> <u>4</u> <u>4</u> <u>3</u>
l	<u>3</u> <u>3</u>	<u>3</u> <u>2</u> <u>4</u> <u>?</u>			
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1</u> <u>1</u>	<u>2</u> <u>2</u>	<u>3</u> <u>3</u>	<u>4</u> <u>4</u>
o	<u>1</u> <u>1</u>	<u>1</u> <u>2</u> <u>2</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>2</u> <u>4</u> <u>3</u> <u>2</u>	<u>4</u> <u>5</u> <u>3</u> <u>3</u>
s	<u>2</u> <u>2</u>	<u>1</u> <u>2</u> <u>3</u> <u>1</u>	<u>2</u> <u>3</u> <u>2</u> <u>2</u>	<u>3</u> <u>3</u> <u>3</u> <u>3</u>	<u>3</u> <u>4</u> <u>4</u> <u>3</u>
l	<u>3</u> <u>3</u>	<u>3</u> <u>2</u> <u>4</u> <u>2</u>			
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 ?</u>		
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> 1	<u>1 2</u> 2 1	<u>2 3</u> 2 2	<u>2 4</u> 3 2	<u>4 5</u> 3 3
s	<u>2</u> 2	<u>1 2</u> 3 1	<u>2 3</u> 2 2	<u>3 3</u> 3 3	<u>3 4</u> 4 3
l	<u>3</u> 3	<u>3 2</u> 4 2	<u>2 3</u> 3 2		
o	<u>4</u> 4				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 ?</u>	
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> 1	<u>1 2</u> 2 1	<u>2 3</u> 2 2	<u>2 4</u> 3 2	<u>4 5</u> 3 3
s	<u>2</u> 2	<u>1 2</u> 3 1	<u>2 3</u> 2 2	<u>3 3</u> 3 3	<u>3 4</u> 4 3
l	<u>3</u> 3	<u>3 2</u> 4 2	<u>2 3</u> 3 2	<u>3 4</u> 3 3	
o	<u>4</u> 4				



		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 ?</u>
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>				

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 ?</u>			

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>			

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 ?</u>		

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>		

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 ?</u>	

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	



		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 ?</u>

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 3</u>

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 3</u>

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 3</u>

How do I read out the editing operations that transform OSLO into SNOW?

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 3</u>

cost	operation	input	output
1	insert	*	w

			s	n	o	w
		0	1 1	2 2	3 3	4 4
o		1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s		2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l		3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o		4 4	4 3 5 3	3 3 4 3	2 4 4 2	4 5 3 3

cost	operation	input	output
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1</u> <u>1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2</u> <u>2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3</u> <u>3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4</u> <u>4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 3</u>

cost	operation	input	output
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	<u>0</u>	<u>1 1</u>	<u>2 2</u>	<u>3 3</u>	<u>4 4</u>
o	<u>1 1</u>	<u>1 2</u> <u>2 1</u>	<u>2 3</u> <u>2 2</u>	<u>2 4</u> <u>3 2</u>	<u>4 5</u> <u>3 3</u>
s	<u>2 2</u>	<u>1 2</u> <u>3 1</u>	<u>2 3</u> <u>2 2</u>	<u>3 3</u> <u>3 3</u>	<u>3 4</u> <u>4 3</u>
l	<u>3 3</u>	<u>3 2</u> <u>4 2</u>	<u>2 3</u> <u>3 2</u>	<u>3 4</u> <u>3 3</u>	<u>4 4</u> <u>4 4</u>
o	<u>4 4</u>	<u>4 3</u> <u>5 3</u>	<u>3 3</u> <u>4 3</u>	<u>2 4</u> <u>4 2</u>	<u>4 5</u> <u>3 3</u>

cost	operation	input	output
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w



		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2	2 4 3 2	4 5 3 3
s	2 2	1 2 3 1	2 3 2 2	3 3 3 3	3 4 4 3
l	3 3	3 2 4 2	2 3 3 2	3 4 3 3	4 4 4 4
o	4 4	4 3 5 3	3 3 4 3	2 4 4 2	4 5 3 3

cost	operation	input	output
1	delete	o	*
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

cost	operation	input	output
1	insert	*	c
1	insert	*	a
1	insert	*	t
0	(copy)	c	c
0	(copy)	a	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

cost	operation	input	output
0	(copy)	c	c
1	insert	*	a
1	insert	*	t
1	insert	*	c
0	(copy)	a	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

cost	operation	input	output
0	(copy)	c	c
0	(copy)	a	a
1	insert	*	t
1	insert	*	c
1	insert	*	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	1 1	2 2	3 3	4 4	5 5	6 6
c	1 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4	6 7 5 5
a	2 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3	5 6 4 4
t	3 3	3 2 4 2	2 1 3 1	0 2 2 0	2 3 1 1	3 4 2 2	3 5 3 3

cost	operation	input	output
0	(copy)	c	c
0	(copy)	a	a
0	(copy)	t	t
1	insert	*	c
1	insert	*	a
1	insert	*	t

# Outline

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

# Spelling correction

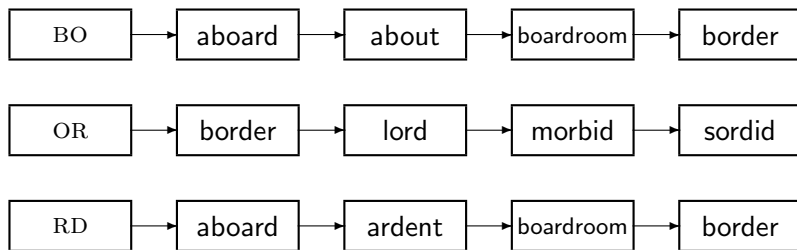
- Isolated spell correction
  - Based on edit distance. How is this done?
  - $k$ -gram indexes. Next few slides.
- Context-sensitive spelling correction
  - Hit-based algorithm
  - Language models



# $k$ -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

- 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.
- More efficient alternative: look at the query log, i.e., a “collection” of queries, not documents
- Even more efficient: build a language model to predict the misspelled word. We will talk about this later in the course.

# 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.

- Nice discussion of Google's spell checker:  
<http://norvig.com/spell-correct.html>
- (Do not read after the "How it Works" title. We will revisit this when we look at language models!)

# 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.
- 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

- ➊ Retain the first letter of the term.
- ➋ Change all occurrences of the following letters to '0' (zero): A, E, I, O, U, H, W, Y
- ➌ 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
- ➍ Repeatedly remove one out of each pair of consecutive identical digits
- ➎ 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*

- Retain H
- *ERMAN* → *ORM0N*
- *ORM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- Return *H655*
- Note: *HERMANN* will generate the same code

# 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