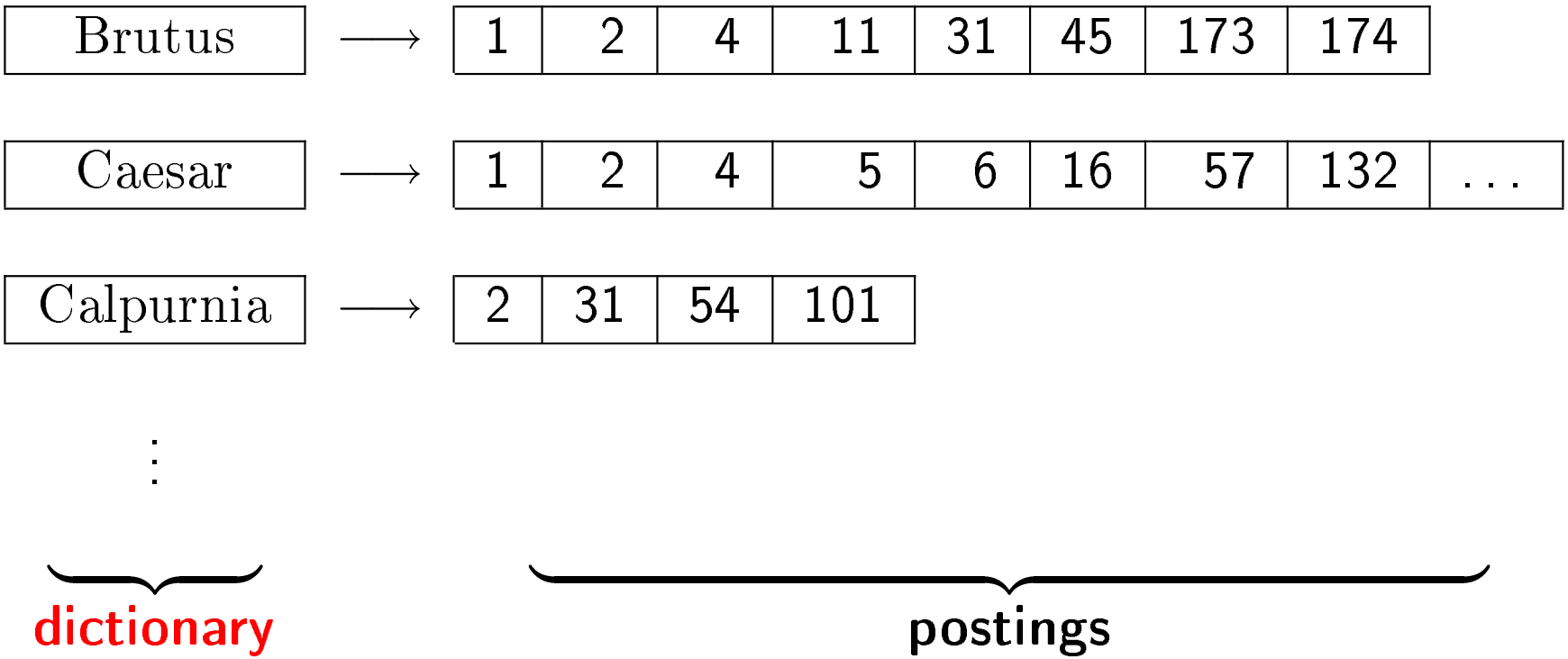
Dictionaries and Tolerant Retrieval [1]

September, 2009

[1] Vorlage: Folien von M. Schütze zu [1]

Inverted index

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



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 an element in this array at query time?

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?

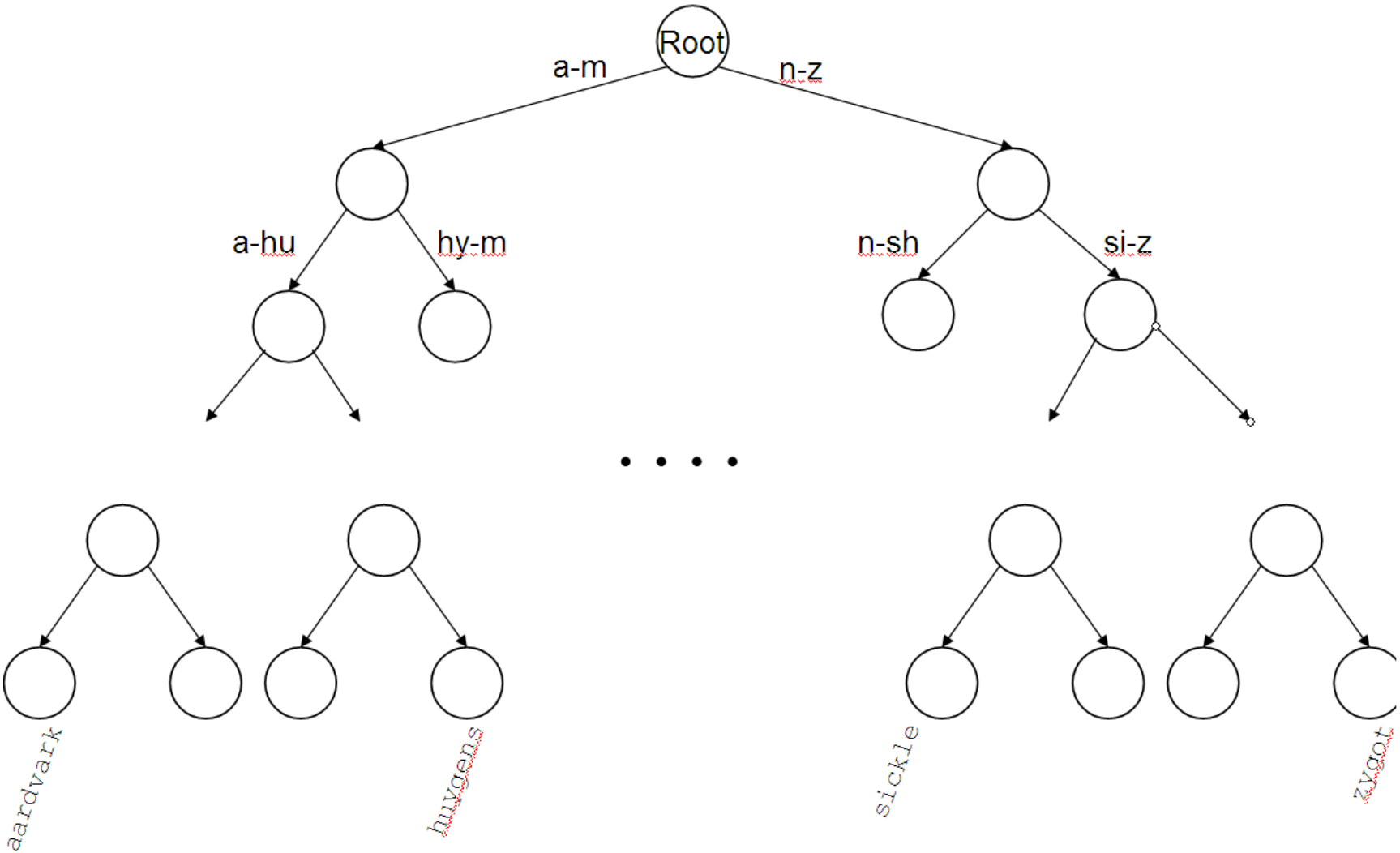
Hashes

- Each vocabulary term is hashed into an integer.
- Try to avoid collisions
- At query time, do the following: hash query term, resolve collisions, 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

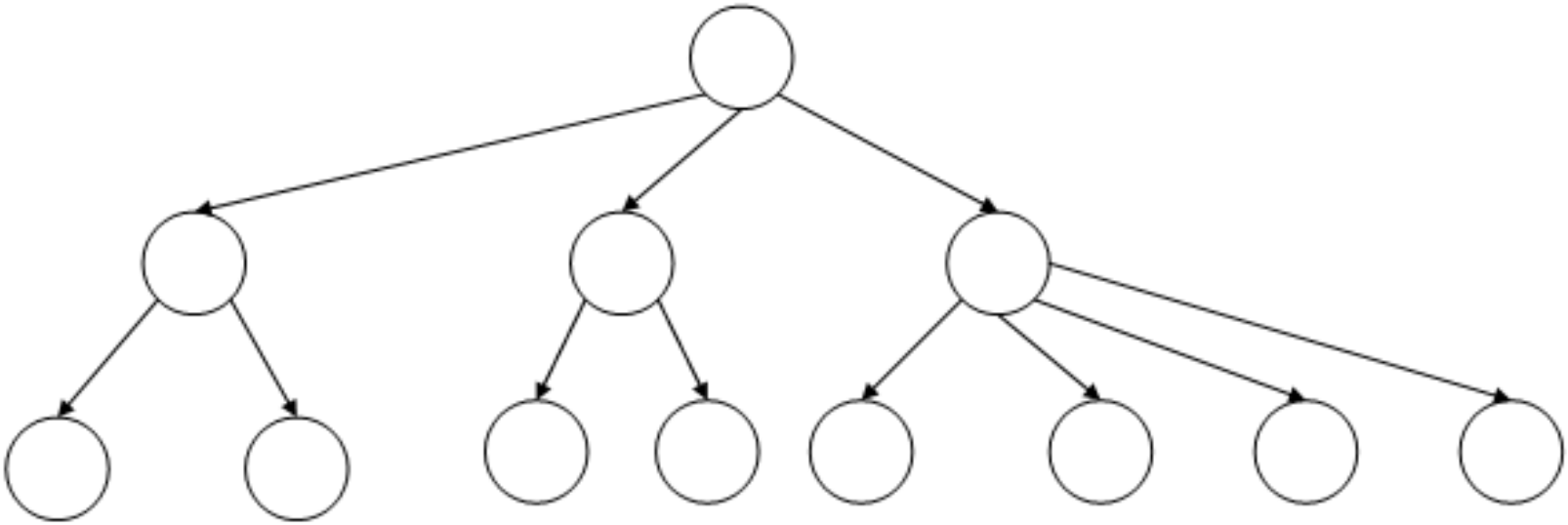
Trees

- 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



Query processing

- At this point, we have an enumeration of all terms in the dictionary that match the wildcard query.
- We still have to look up the postings for each enumerated term.
- E.g., consider the query: *gen** and *universit**
- This may result in the execution of many Boolean and queries.
- If there are 100 terms matching *gen** and 50 terms matching *universit** , then we have to run 5000 Boolean queries!

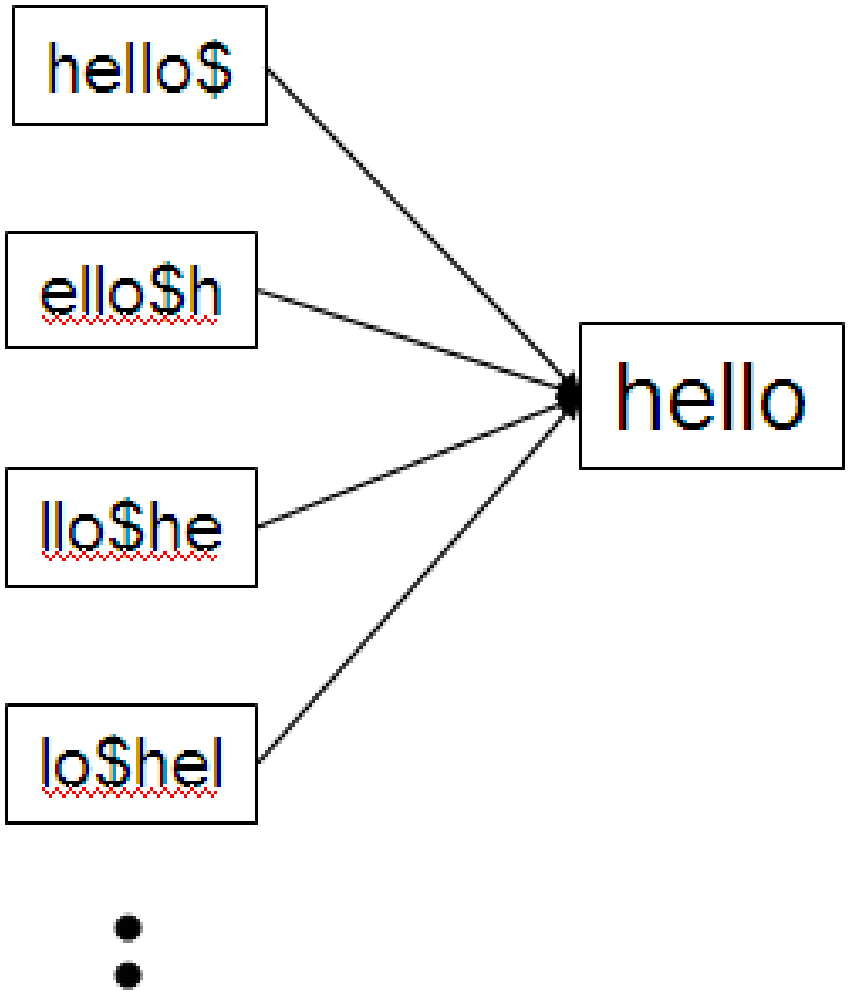
How to handle * in the middle of a term

- 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.

Permuterm index

- For term hello: add hello$, ello$h, llohe, lohel, and o$hell to the B-tree where $ is a special symbol

Permuterm → term mapping



Permuterm index

- For hello, we've stored: hello$, ello$h, llohe, lohel, and o$hell
- Queries
  - For X, look up X$
  - 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*
- It's really a tree and should be called permuterm tree.
- But permuterm index is 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.
- Maintain an inverted index from bigrams to the terms that contain the bigram

Postings list in a 3-gram inverted index

Bigram 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 indexes are more space efficient than permuterm indexes.

Processing wildcard queries in the term-document index

- As before, we must potentially execute a large number of Boolean queries
- Most straightforward semantics: Conjunction of disjunctions
- Recall the query: *gen** and *universit**
  - (*geneva* and *university*) or (*geneva* and *université*) or (*genève* and *university*) or (*genève* and *université*) or (*general* and *universities*) or .....
- Very expensive
- Does Google allow wildcard queries?
- Users hate to type.
- If abbreviated queries like [pyth* theo*] for *pythagoras' theorem* are legal, users will use them ...
  - ... a lot
- According to Google search basics, 2009.04.27: "Note that the * operator works only on whole words, not parts of words."
- But that does not seem to be true. Try [pythag*] and [m*nchen]

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.
- 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
- We can use the term vocabulary of the inverted index as the list of correct words.
- 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 ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7      do if s1[i] = s2[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[|s1|, |s2|]
```

Operations: insert, delete, replace, copy

Levenshtein distance: Example

		f		a		s		t	
		0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
c	**1**	*1*	*2*	**2**	3	**3**	4	**4**	5
	1	*2*	*1*	**2**	**2**	**3**	**3**	**4**	**4**
a	**2**	**2**	**2**	*1*	*3*	*3*	*4*	4	5
	2	3	**2**	*3*	*1*	*2*	*2*	**3**	**3**
t	**3**	**3**	**3**	3	**2**	*2*	*3*	*2*	*4*
	3	4	**3**	4	**2**	*3*	*2*	*3*	*2*
s	**4**	**4**	**4**	4	**3**	**2**	3	*3*	*3*
	4	5	**4**	5	**3**	4	**2**	*3*	*3*

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

http://ifnlp.org/lehre/teaching/2008-SS/ir/editdist2.pdf

Exercise

- Given: cat and catcat
- Compute the matrix of Levenshtein distances
- Read out the editing operations that transform cat into catcat

Dynamic programming (Cormen et al.)

- Optimal substructure: The optimal solution to the problem contains within it optimal solutions to subproblems.
- Overlapping subproblems: The optimal solutions to subproblems ("subsolutions") overlap. These subsolutions are computed over and over again when computing the global optimal solution.
- Optimal substructure: We compute minimum distance of substrings in order to compute the minimum distance of the entire string.
- Overlapping subproblems: Need most distances of substrings 3 times (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.
- Or do automatic correction – but this is potentially expensive and disempowers the user.

Example 1

- Levenshtein distance oslo – snow?

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

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	1 2 / 2 ?			
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

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

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	2 3 / 2 ?		
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**		
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** **1**	**1** 2	2 **1**	**2** **2**	3 **2**	2 3	4 ?		
s	**2** **2**								
l	**3** **3**								
o	**4** **4**								

		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**	
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** / 2 2 / **1**	**2** / **2** 3 / **2**	**2** / 3 4 / **2**	4 / 3 5 / ?
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

		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**				
l	**3** / **3**				
o	**4** / **4**				

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** / **1**	**1** / 2	2 / **1**	**2** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3 / **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 / **3**	5 / 3

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

How do I read out the editing operations that transform oslo into snow?

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

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** / **2**	3 / **2**	**2** / 3	4 / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	3 / **2**	**3** / **3**	3 / **3**	**3** / 4	4 / **3**
l	**3** / **3**	3 / 4	**2** / **2**	**2** / 3	3 / **2**	**3** / **3**	4 / **3**	**4** / **4**	4 / **4**
o	**4** / **4**	4 / 5	**3** / **3**	**3** / 4	3 / **3**	2 / 4	4 / **2**	4 / **3**	5 / **3**

cost	operation	input	output
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	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** / **2**	**3** / **2**	**2** / 3	**4** / **2**	4 / **3**	5 / **3**
s	**2** / **2**	**1** / 3	2 / **1**	**2** / **2**	**3** / **2**	**3** / **3**	**3** / **3**	**3** / 4	4 / **3**
l	**3** / **3**	3 / 4	**2** / **2**	**2** / 3	3 / **2**	**3** / **3**	4 / **3**	**4** / **4**	4 / **4**
o	**4** / **4**	4 / 5	**3** / **3**	**3** / 4	**3** / **3**	**2** / 4	**4** / **2**	4 / **3**	5 / **3**

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

Example 2

- Levenshtein distance cat – catcat: 3
- How many distinct minimal sequences of editing operations?

		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

Effiziente Vergleiche

- Aufwand der Berechnung für Edit-Distance ist $O(|string1| * |string2|)$
- Problemstellung: Alle Terme aus einem Dictionary finden, die einen Abstand d kleiner/gleich einem Maximalabstand haben
- Wie können wir vermeiden, dass wir bei der Spell-Korrektur zu allen Query-Term Paaren die Berechnung durchführen müssen?
- Beachte: Edit-Distanz ist eine (diskrete) Metrik
- Um die Edit-Distance nicht für alle Query-Term Paare berechnen zu müssen, kann man die Metrik-Eigenschaft in Kombination mit einer geeigneten Datenstruktur (spezieller Baum) benutzen.

Effiziente Such-Bäume für metrische Abstände

Definition einer Metrik:

- **Non negativity**
  $\forall s, t \in \mathbb{U}: \quad d(s,t) \geq 0$
- **Symmetry**
  $\forall s, t \in \mathbb{U}: \quad d(s,t) = d(t,s)$
- **Identity**
  $d(s,t) = 0 \Leftrightarrow s = t$
- **Triangle inequality**
  $\forall r, s, t \in \mathbb{U}: \quad d(r,t) \leq d(r,s) + d(s,t)$

Burkhard-Keller Tree (BKT) - Konstruktion

Eine Datenstruktur, die den Suchraum durch Benutzen der (diskreten) Metrik einschränkt, ist der Burkhard-Keller Tree (siehe [2])

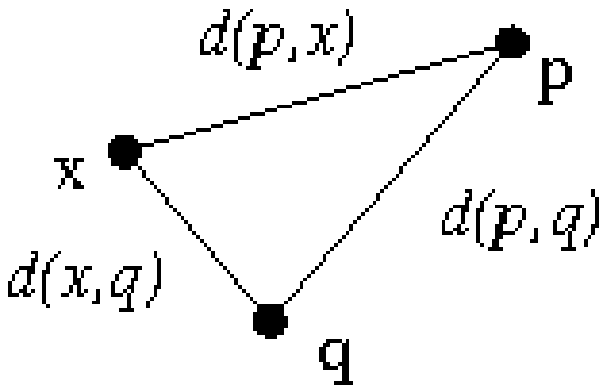
1. Nehme ein beliebiges Element der Menge $\mathbb{U}$ als (Root-)Node
2. Partitioniere die restlichen Elemente in Gruppen, nach dem (diskreten) Abstand i zum ausgewählten (Root-)Element
3. Die Gruppen werden zu Unterbäumen zum ausgewählten Element. Baue die Unterbäume rekursiv auf (gehe zu 1), bis die (Unter-)Menge nur noch aus einem Element besteht.

Burkhard-Keller Tree - Suche

Idee: Benutze (indirekt) die Dreiecksungleichung, um die Anzahl der zu besuchenden Knoten zu beschränken. Ziel: Alle Elemente x zu finden, mit dem Abstand $d(q, x) \leq r$; q: Query-Element

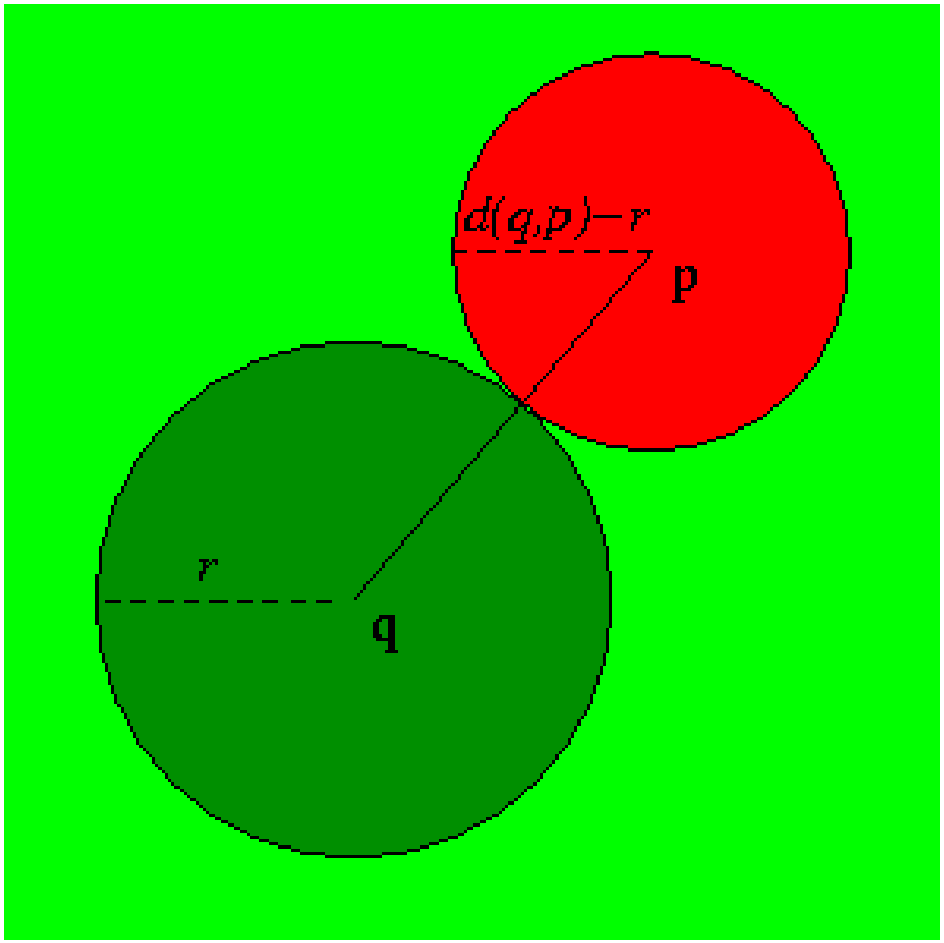
1. Berechne den Abstand des Query-Elements q zum (Root)-Node Element p
2. Falls der Abstand $d(p, q)$ zwischen der "Node" p und dem Suchstring q kleiner-gleich als ein Mindestabstand r ist $(d(p, q) \leq r)$, füge den Inhalt zu Ergebnissmenge hinzu
3. Steige rekursiv **nur** in alle Unterzweige ab, die folgende Ungleichung erfüllen (Erklärung folgt): $\max(d(q, p) - r, 0) \leq i \leq d(p, q) + r$
4. Beachte: Für alle abgestiegenen Knoten muss wieder der Abstand zum Knoten (neues p) berechnet werden

Burkhard-Keller Tree - Min-Bedingung



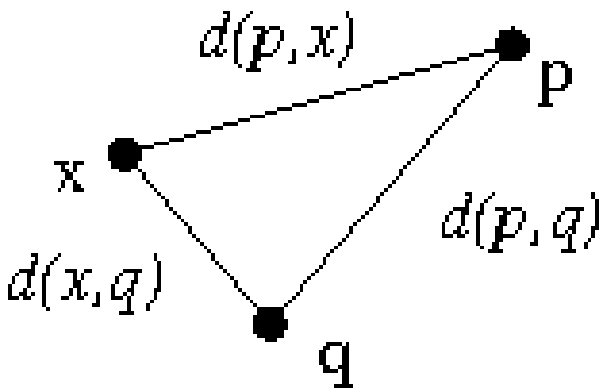
- $d(p, q) \leq d(p, x) + d(x, q)$
- gesuchten x_g müssen folgende Bedingung erfüllen: $d(x_g, q) \leq r$
- $d(p, q) \leq d(p, x_g) + r$
- $d(p, q) - r \leq d(p, x_g)$
- Mit **non negativity** und i für $d(p, x_g)$:
- $\max(d(q, p) - r, 0) \leq i$

BKT - graphischen Darstellung der Min-Bedingung



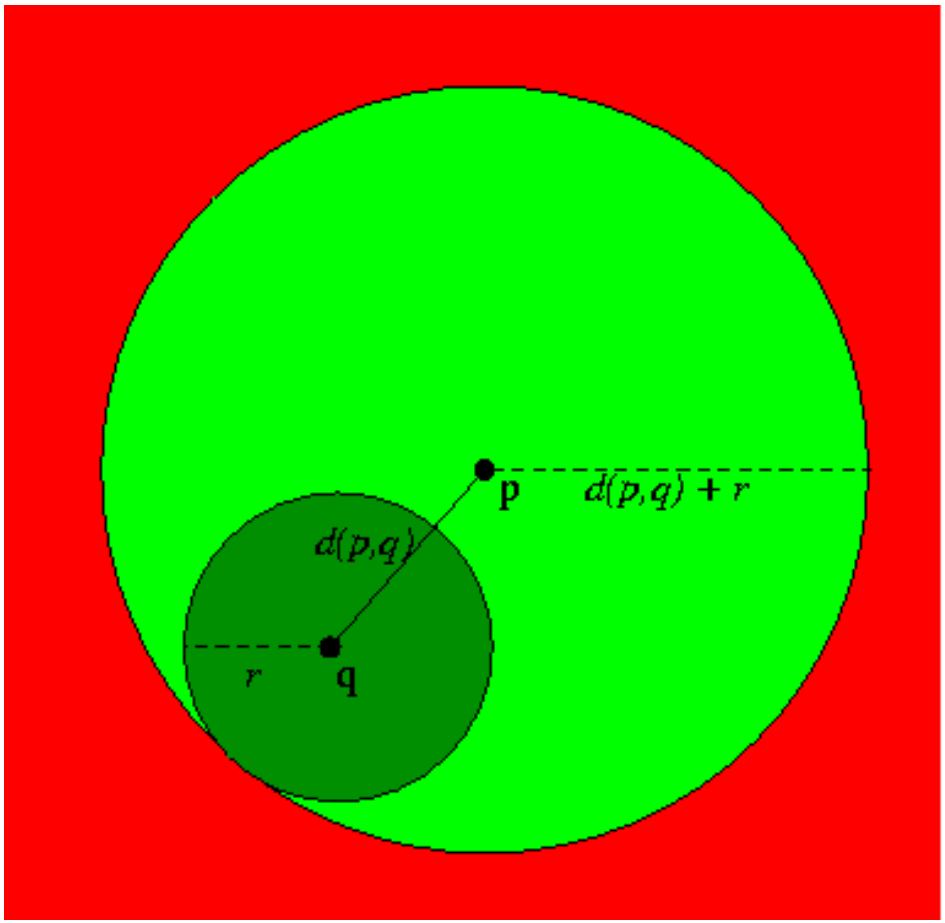
- rot: verbotene x nach der Min-Bedingung $\max(d(q,p)-r,0) \leq d(x,q)$
- hellgrün: Erlaubte x nach der Min-Bedingung
- dunkelgrün: x mit $d(x,q) \leq r$

BKT: Max-Bedingung



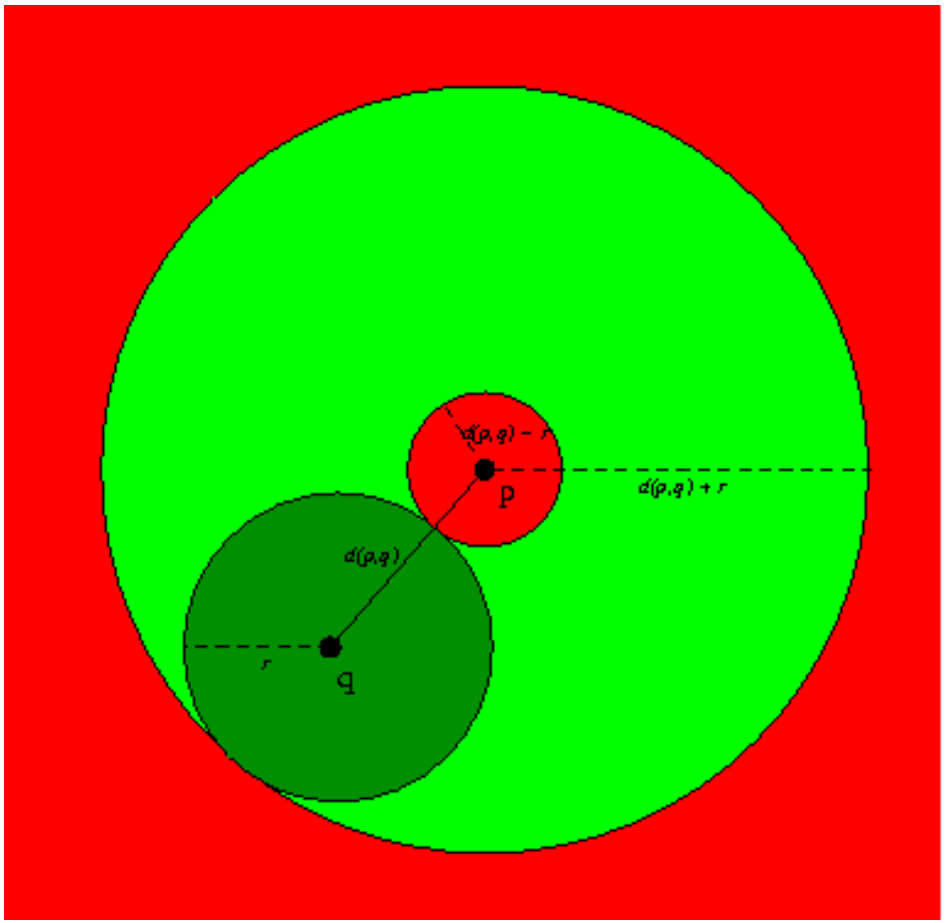
- aus $d(p,x) \leq d(p,q) + d(q,x)$
- Bedingung für die gesuchten x_g: $d(q,x_g) \leq r$
- $d(p,x_g) \leq d(p,q) + r$
- i entspricht $d(p,x_g)$
- $i \leq d(p,q) + r$

BKT - graphischen Darstellung der Max-Bedingung



- rot: Verbotene x nach der Max-Bedingung $d(x, q) \leq d(p, q) + r$
- hellgrün: Erlaubte x nach der Max-Bedingung
- dunkelgrün: x mit $d(x, q) \leq r$

Burkhard-Keller Tree - Ungleichung

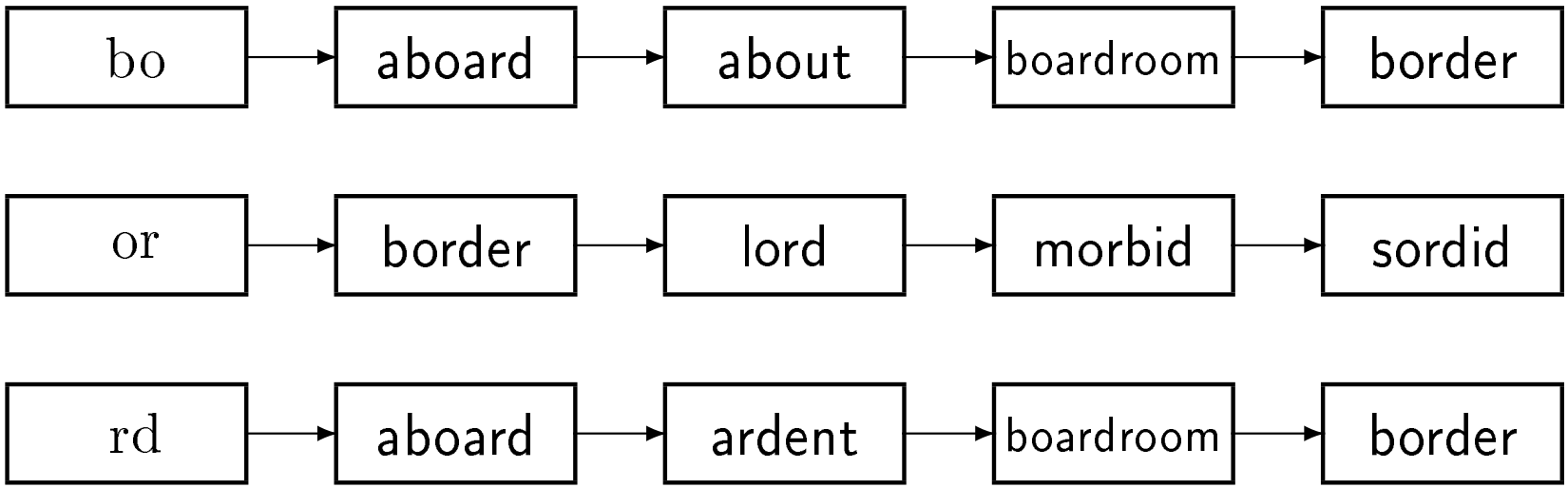


- rot: Verbotene x
- hellgrün: Erlaubte x nach der Ungleichung: $\max(d(q,p) - r, 0) \leq i(= d(x,q)) \leq d(p,q) + r$
- dunkelgrün: x mit $d(x,q) \leq r$

k-gram indexes for spelling correction

- Enumerate all k-grams in the query term
- 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
- Example: bigram index, misspelled word bordroom
- Bigrams: bo, or, rd, dr, ro, oo, om

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, 19 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 "collection" of queries, not 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.

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

- Retain H
- ERMAN → 0RM0N
- 0RM0N → 06505
- 06505 → 06505
- 06505 → 655
- Return H655
- Will HERMANN 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.

Resources

- Chapter 3 of IIR
- Resources at http://ifnlp.org/ir
  - Soundex demo
  - Levenshtein distance demo
  - Levenshtein distance slides
  - Peter Norvig's spelling corrector


H. S. Christopher Manning, P. Raghavan.
Introduction to Information Retrieval.
Cambridge, 2008.


P. Zezula, G. Amato, V. Dohnal, and M. Batko.
Similarity Search: The Metric Space Approach (Advances in Database Systems).
Springer, November 2005.