BITS Pilani
Pilani Campus

Information Retrieval

Abhishek
January 2020

BITS Pilani
Pilani Campus

CS F469, Information Retrieval
Lecture No. 5

Recap of Lecture 4

- Faster merging of posting lists
- Positional postings and phrase queries
- Collocations

Today's Lecture

- Tolerant Retrieval
  - Wildcard queries
  - Spelling correction

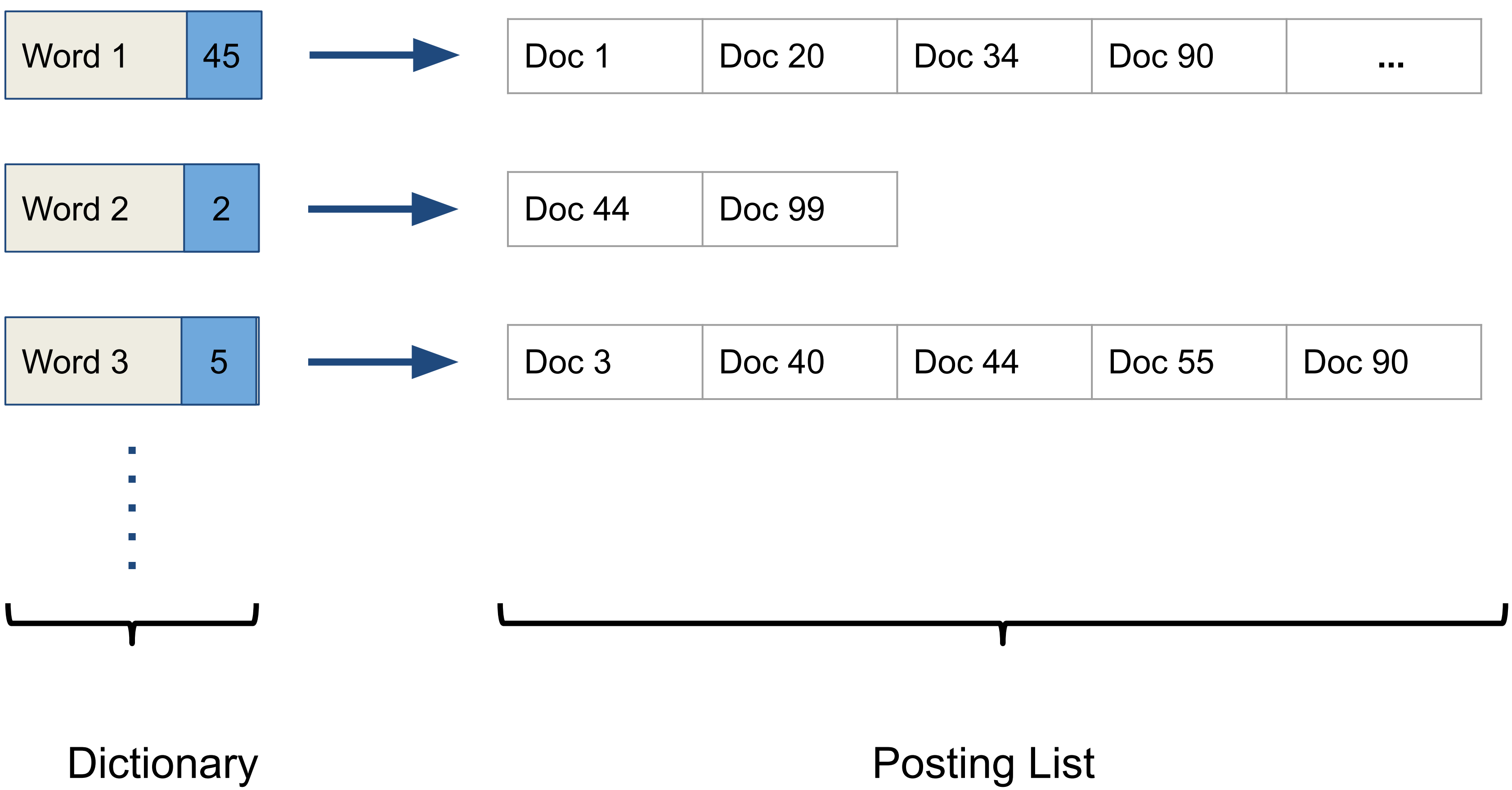
Tolerant Retrieval

Objective: To develop techniques that are robust to typographical errors in query, as well as alternate spellings.

Why?

- User might be uncertain about how to spell a query term.
- User might be seeking for documents containing variants of query terms. (automatic, automation, automated)
- User can make spelling errors. (cherrapunji vs cherapunji)

Recap: Inverted index using Dictionaries



Dictionaries

The dictionary is a data structure used for storing the term vocabulary.

key : value

key = term

value = term frequency, pointer to posting list, and some other information if required.

How do we perform Term Lookup in Dictionaries?

- Two categories of solutions:
  - Hash tables
  - Trees

How do we perform Term Lookup in Dictionaries?

- Two categories of solutions:
  - Hash tables
  - Trees

- How should we decide?
  - How many keys are we likely to have?
  - Is the number likely to remain static or change a lot?
  - In the case of changes, are we likely to have only new keys inserted or some keys in the dictionary can be deleted?
  - What are relative frequencies with which various keys will be accessed?

Hashes

- Every key will be hashed to a integer.
- At query time: Find hashes of every term and fetch the corresponding data (posting lists, etc.).

Pros:

- Lookup time is constant

Cons:

- No way to find minor variants (naive vs naïve)
- No prefix search (all words starting with automat)
- Periodically rehashing will be required, if vocabulary keeps on growing.

Trees

- Tree solves prefix search issue.
- Efficient search, O(log(M)), where M is the size of vocabulary. (but slower than hash tables)
- O(log(M)) only holds for balanced trees.
- Balancing in binary trees can be expensive.

Example Tree Constructed for String Search

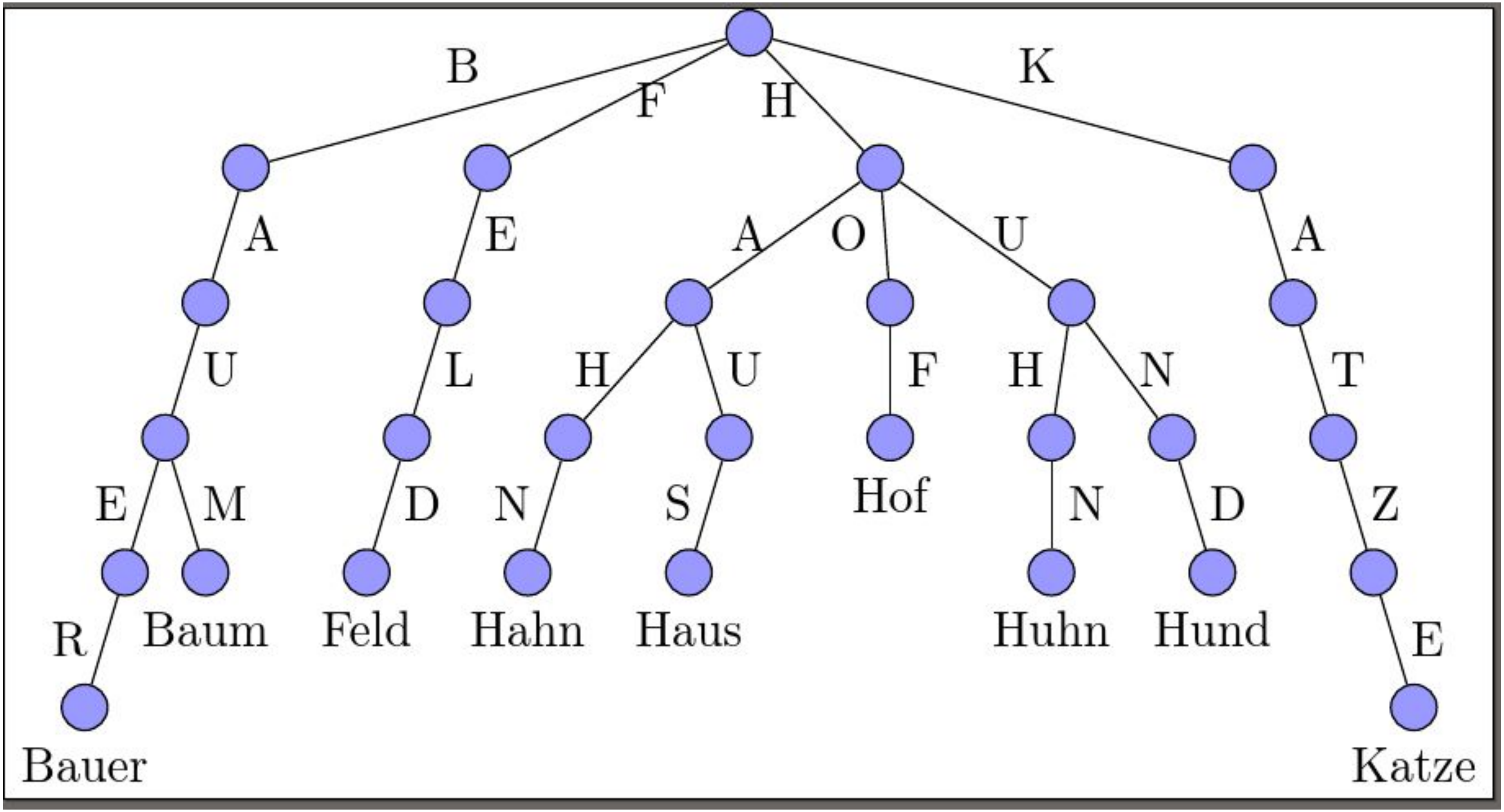


Image source: https://tex.stackexchange.com/questions/182460/creating-trie-trees-in-tikz

Example Tree Constructed for String Search

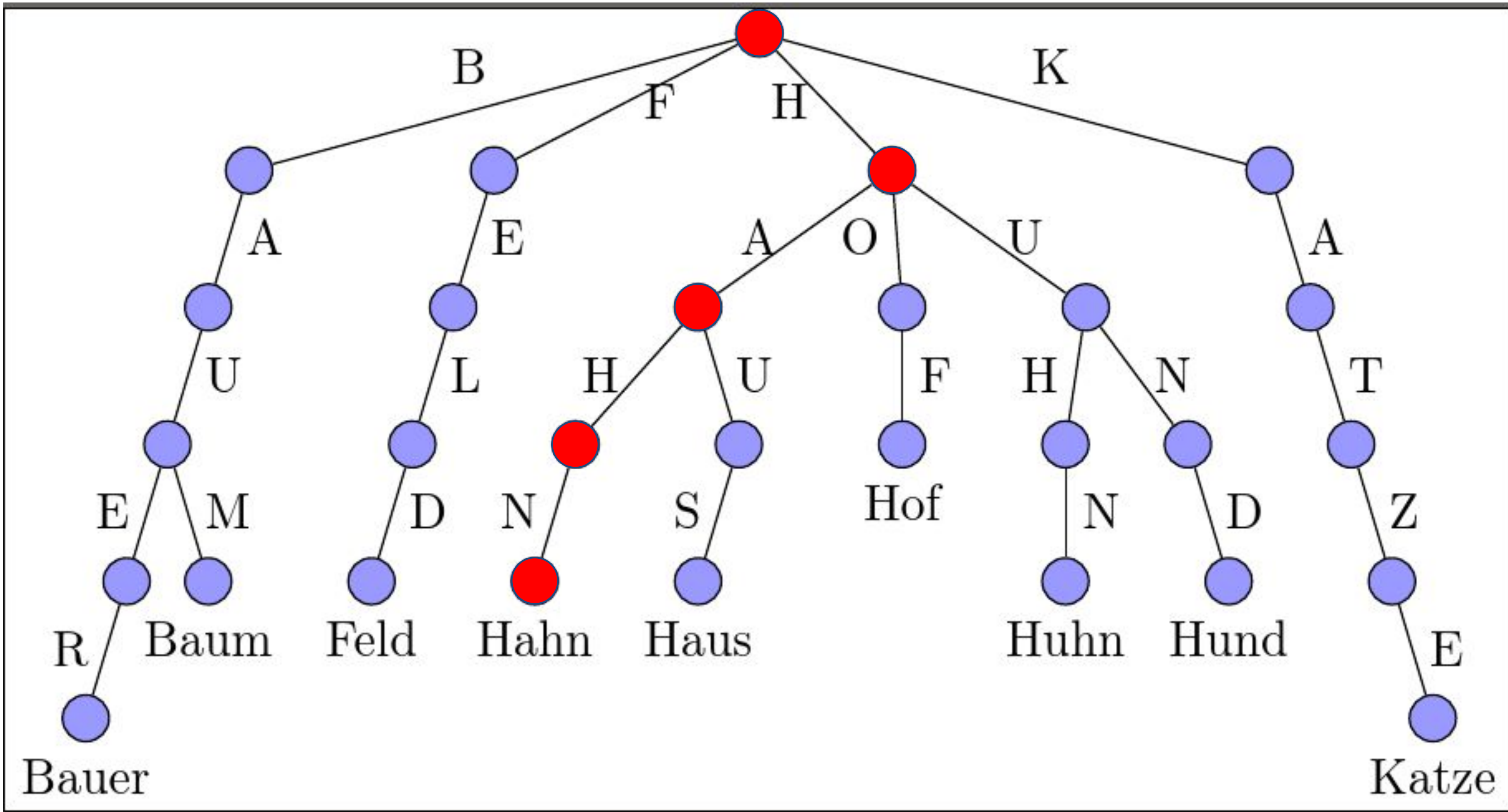
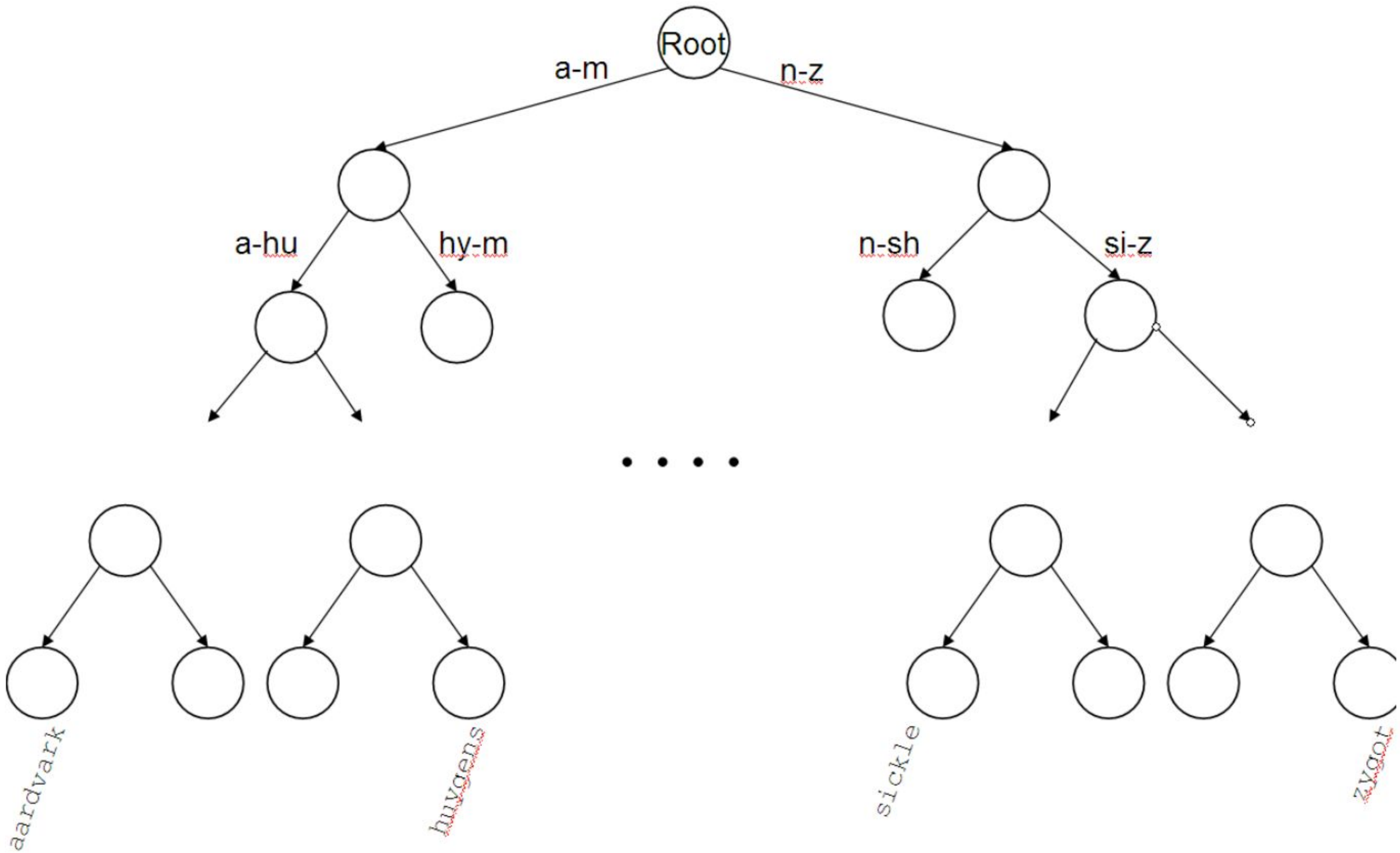


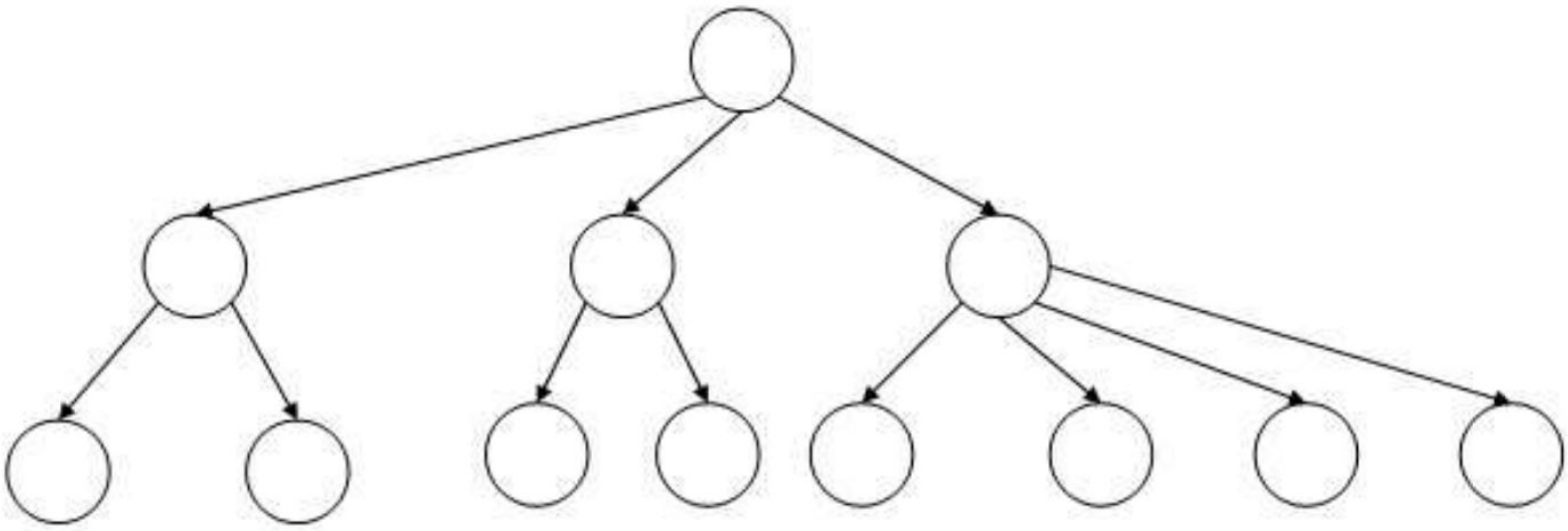
Image source: https://tex.stackexchange.com/questions/182460/creating-trie-trees-in-tikz

Binary Trees



Example from Figure 3.1, Introduction to IR, C.D. Manning, P. Raghavan and H. Schütze.

B-Trees



Example from Figure 3.2, Introduction to IR, C.D. Manning, P. Raghavan and H. Schütze.

B-Trees

- **B-tree definition:** Every internal node has a number of children in the interval [a, b] where a, b are appropriate positive integers or characters, e.g., [2, 4]
- Self balancing
- I/O efficient.

Wildcard Queries

Wildcard Queries

Queries of the form:

- **automat*** : This can be expanded to automatic, automated, automation, etc.
- ***er** : This can be expanded to writer, singer, worker, etc.
- **a*e*i*o*u** : Any term that include all vowels in sequence.

Some use cases for wildcard queries

- The user is uncertain about the correct spelling of a query term (e.g., Sidney vs Sydney → S*dney).
- The user is aware of multiple spelling and (consciously) seeks documents containing the variants (e.g., color vs colour)
- The user seeks documents containing variants of terms that might be caught by stemming (e.g., judicial vs judiciary).
- The user is uncertain about the correct rendition of a foreign word (e.g., Universit* Stuttgart) (German: Universität)

Using Trees for Wildcard Query Search

- **Prefix search?** (Queries of the form **red***)

Using Trees for Wildcard Query Search

- **Prefix search?** (Queries of the form **red***)
  - Easy for B-trees.
  - Example query: **red***
    - Retrieve all terms **t** in the range **red ≤ t < ree**
  - Example query: **mon***
    - Retrieve all terms **t** in the range **mon ≤ t < moo**
- **Result:** Set of terms that satisfy the wildcard.
- Then return documents that contains any of these terms.

Using Trees for Wildcard Query Search

- **Suffix search?** (Queries of the form ***er**)

Using Trees for Wildcard Query Search

- **Suffix search?** (Queries of the form ***er**)
- Reverse B-tree
- Construct a B-tree constructed on the reversed terms.
- For example, the term **singer** in the reverse B-Tree would have the following path:

$$\text{root} \rightarrow r \rightarrow e \rightarrow g \rightarrow n \rightarrow i \rightarrow s$$

- Then search for the prefix **re***

Using Trees for Wildcard Query Search

- **Single * ?**
  - Example: **a*e**

Using Trees for Wildcard Query Search

- **Single * ?**
  - Example: **a*e**
  - Can use combination of prefix and suffix search:
    - Find terms for **a*** and ***e**, separately.
    - Perform an intersection of retrieved terms.
    - Retrieve documents using the intersection terms.

General Wildcard Queries

- Permuterm indexes
- K-gram indexes

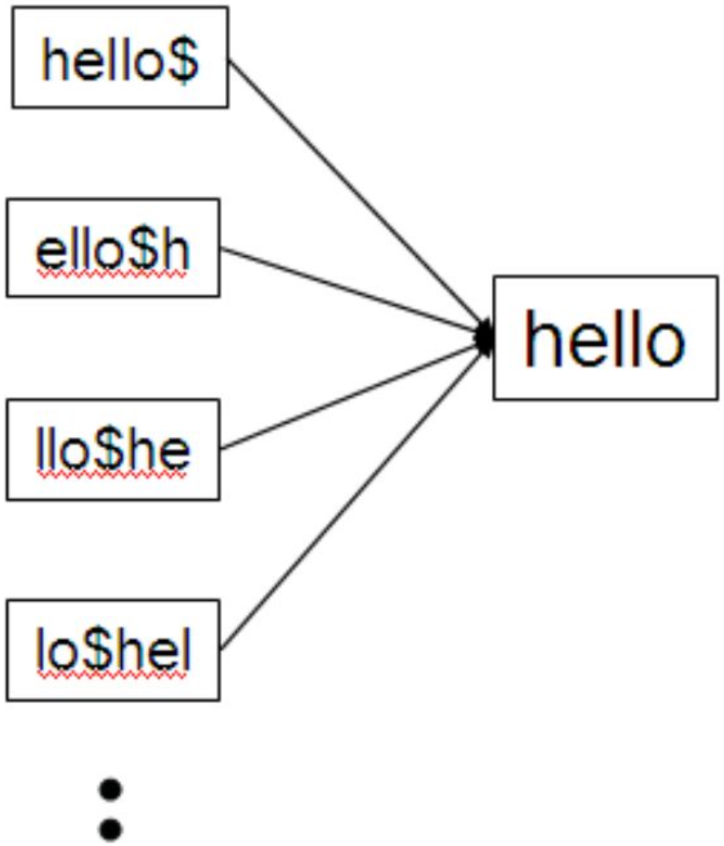
Permuterm Indexes

Permuterm Indexes

- Create a new index (permuterm index, or permuterm tree).
- Example for term: **hello**
- Add **hello$**, **$hello**, **o$hell**, **lo$hel**, **llo$he**, **ello$h** to the new index (tree).
- $ is a boundary symbol appended to the end of the term.

Permuterm Indexes

permuterm → term mapping



Wildcard Searches in Permuterm Indexes

- **Query:** m*n
- Add boundary in the end and rotate the query such that * appears in the end.
- After rotation: n$m*
- Lookup for term n$m* in the permuterm index.
- The lookup should match: **man** and **moron** as they will have a rotated entry of **n$ma** and **n$moro**.

Wildcard Searches in Permuterm Indexes

- **Query:** fi*mo*er
- **Idea:** Add boundary, rotate, and try to search for longest prefixes or suffixes.
- Lookup for term **er$fi*** in the permuterm index.
- Enumerate the resulting terms exhaustively, checking each terms if it contains **mo**.
- Terms that survived after filtering are then searched in the standard inverted index.
- For the above example, the term **fishmonger** would survive the filtering but **filibuster** would not.

Permuterm Indexes

- **Disadvantage:**
  - The dictionary becomes very larges as it contains all rotations of the terms.

K-gram indexes

K-gram indexes

- Enumerate all k-gram characters of the term.
- 2-grams are called bigrams and 3-grams are called trigrams.
- Example term: hello
  - 2-grams: h, he, el, ll, lo, l
  - 3-grams: he, hel, ell, llo, lo
- $ is a boundary symbol, added before and after the term.
- Make an inverted index from k-grams to the terms that contains that k-gram.

Example K-gram Inverted Index

K-gram		Terms			
etr	→	beetroot	metric	pertify	retrieval
red	→	red	reddish		
er$	→	writer	singer	worker	partner

Note: We have now two types of inverted indexes.

1. Term document inverted index.
2. K-gram term inverted index.

Wildcard Searches using K-gram Index

- Example Query: **red***
- 3-grams: **$re**, **red**
- First use the boolean query: **$re AND red** to the 3-gram index.
- The retrieved terms will be ***red***, ***retired***, ***reddish***, etc.
- Perform a post filtering step.
- Terms that survived after filtering are then searched in the standard inverted index.

K-gram indexes

- **Advantage:**
  - Space efficient as compared to permuterm

Issues with Boolean Wildcard Queries

- **Expensive query execution:**
  - A* Sm* $\rightarrow$ (disjunctions) AND (disjunctions)
  - Lots of possible options:
    - Alan Smith
    - Alan Smales
    - Alex Smith
- Encourages laziness

Spelling Correction

Spelling Correction

- Two basic principles underlying most spelling correction algorithms are:

  1. Of various alternate spellings for a misspelled query, choose the “nearest” one.

  2. When there are multiple correctly spelled queries with same (or nearly same) proximity score, select the one that is most common (eg. ***grunt*** and ***grant*** are possible correction for ***grnt***).

Few design decisions related to Spelling Corrections

1. For a misspelled query always retrieve documents for the original query as well as spell-corrected variant of query (e.g., for query **carot**, retrieve documents containing **carot**, **carrot**, and **tarot**).
2. As in (1) above, but only when query term **carot** is not in the dictionary.
3. As in (1) above, but only when the original query returned fewer than a preset number of documents.
4. When the original query returned fewer documents, prompt user for spelling suggestions.

Forms of Spelling Correction

- Isolated term correction:
  - Correct single term of a query at a time, even if the query is a multi word query.
  - Fails to detect some contextual errors:
    - “flew <u>form</u> Delhi”
- Context-sensitive correction:
  - The surrounding terms around a term are taken into account while suggesting for correction.

String Proximity

Edit distance

- The edit distance between a string s_1 and string s_2 is the minimum number of basic operations that convert s_1 to s_2.
- **Levenshtein distance:** The allowed basic operations are: add, delete and replace.
  - **Examples:**
  - Levenshtein distance **dog-do**: 1
  - Levenshtein distance **cat-cart**: 1
  - Levenshtein distance **cat-cut**: 1
  - Levenshtein distance **cat-act**: 2

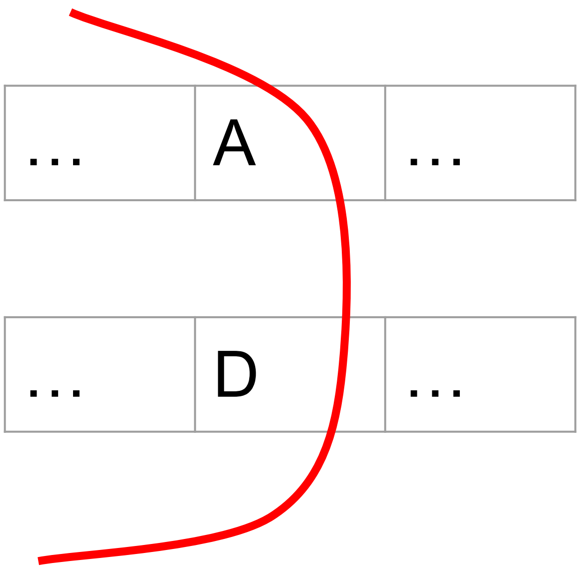
Levenshtein Distance Computation: Dynamic Programming

- **Idea:** Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

Levenshtein Distance Computation: Dynamic Programming

- **Idea:** Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

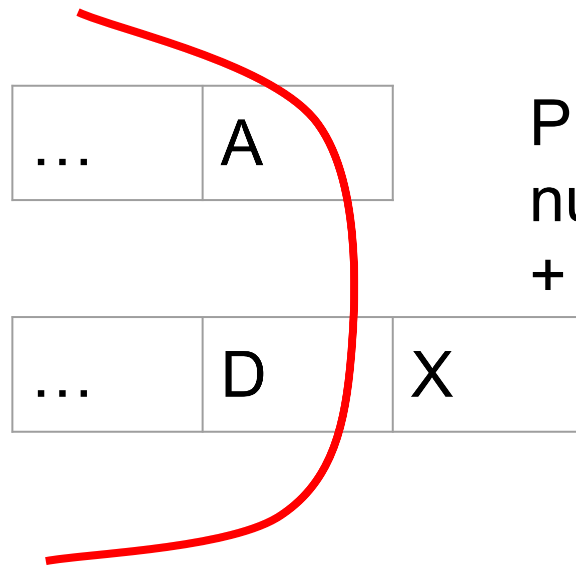
Use the optimal solution for the sub-problems to obtain the final solution.



Levenshtein Distance Computation: Dynamic Programming

- **Idea:** Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

Use the optimal solution for the sub-problems to obtain the final solution.

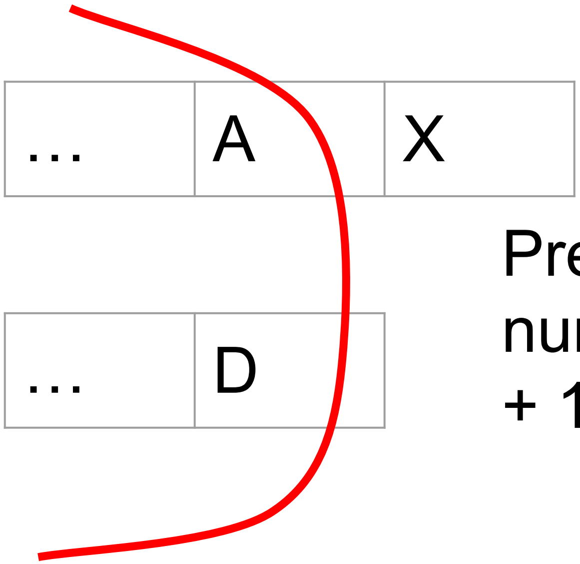


Previous number of edits + 1 addition

Levenshtein Distance Computation: Dynamic Programming

- **Idea:** Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

Use the optimal solution for the sub-problems to obtain the final solution.



Previous number of edits + 1 deletion

Levenshtein Distance Computation: Dynamic Programming

- **Idea:** Simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

Use the optimal solution for the sub-problems to obtain the final solution.

…	A	X

…	D	Y

Previous number of edits + (either replace or copy)

Levenshtein Distance Computation: Example

- String 1: SNOW
- String 2: OSLO

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	S	N	O	W
#					
O					
S					
L					
O					

Levenshtein Distance Computation: Example

Cost of getting here from upper left neighbour (copy or replace)	Cost of getting here from the top (delete)
Cost of getting here from the left (insert)	Minimum of the other three cells

Levenshtein Distance Computation: Example

S_2 (output)

	#	S	N	O	W
#	0				
O					
S					
L					
O					

S_1 (input)

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	S	N	O	W
#	0				
O	1 1				
S	2 2				
L	3 3				
O	4 4				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S$_2$ (output)

S$_1$ (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S$_2$ (output)

S$_1$ (input)

	#	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				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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**			
L	3 3				
O	4 4				

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S_2 (output)

S_1 (input)

	#	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**

Co/R	Del
Ins	min

Levenshtein Distance Computation: Example

S$_2$ (output)

S$_1$ (input)

	#	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**

Co/R	Del
Ins	min

Answer

Levenshtein Distance Computation: Finding Edits

Levenshtein Distance Computation: Finding Edits

S_2 (output)

S_1 (input)

	#	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**

Co/R	Del
Ins	min

Levenshtein Distance Computation: Finding Edits

S_2 (output)

S_1 (input)

	#	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**

Co/R	Del
Ins	min

Cost	Op.	i/p	o/p
1	Ins	*	W

Levenshtein Distance Computation: Finding Edits

S$_2$ (output)

S$_1$ (input)

	#	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**

Co/R	Del
Ins	min

Cost	Op.	i/p	o/p
1	Ins	*	W
0	Co	O	O

Levenshtein Distance Computation: Finding Edits

S_2 (output)

S_1 (input)

	#	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

Co/R	Del
Ins	min

Cost	Op.	i/p	o/p
1	Ins	*	W
0	Co	O	O
1	R	L	N

Levenshtein Distance Computation: Finding Edits

S_2 (output)

S_1 (input)

	#	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

Co/R	Del
Ins	min

Cost	Op.	i/p	o/p
1	Ins	*	W
0	Co	O	O
1	R	L	N
0	Co	S	S

Levenshtein Distance Computation: Finding Edits

S$_2$ (output)

S$_1$ (input)

	#	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

Co/R	Del
Ins	min

Cost	Op.	i/p	o/p
1	Ins	*	W
0	Co	O	O
1	R	L	N
0	Co	S	S
1	Del	O	*

Levenshtein Distance Computation: Finding Edits

Co/R	Del
Ins	min

S_2 (output)

S_1 (input)

	#	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	Op.	i/p	o/p
1	**Ins**	*	W
0	**Co**	O	O
1	**R**	L	N
0	**Co**	S	S
1	**Del**	O	*

Levenshtein Distance Computation: Dynamic Programming

- Can be computed in $O(|s_1| \times |s_2|)$ time.
- The algorithm fills a matrix of dimensions $|s_1| \times |s_2|$.
- $m[i, j]$ = The edit distance between the string constituting of the first **i** characters of s_1 and the first **j** characters of s_2.

Levenshtein Distance Computation: Dynamic Programming

$\textsc{EditDistance}(s_1, s_2)$

$\quad int\ m[i,j] = 0$
$\quad \textbf{for}\ i \leftarrow 1\ \textbf{to}\ |s_1|$
$\quad \textbf{do}\ m[i,0] = i$
$\quad \textbf{for}\ j \leftarrow 1\ \textbf{to}\ |s_2|$
$\quad \textbf{do}\ m[0,j] = j$
$\quad \textbf{for}\ i \leftarrow 1\ \textbf{to}\ |s_1|$
$\quad \textbf{do for}\ j \leftarrow 1\ \textbf{to}\ |s_2|$
$\qquad \textbf{do}\ m[i,j] = \min\{m[i-1,j-1] + \text{if}\ (s_1[i] = s_2[j])\ \text{then}\ 0\ \text{else}\ 1\text{fi},$
$\qquad\qquad\qquad m[i-1,j] + 1,$
$\qquad\qquad\qquad m[i,j-1] + 1\}$
$\quad \textbf{return}\ m[|s_1|, |s_2|]$

Example from Figure 3.5, Introduction to IR, C.D. Manning, P. Raghavan and H. Schütze.

Levenshtein Distance Computation: Dynamic Programming

$\text{EDITDISTANCE}(s_1, s_2)$

$int\ m[i,j] = 0$
for $i \leftarrow 1$ **to** $|s_1|$
do $m[i,0] = i$
for $j \leftarrow 1$ **to** $|s_2|$
do $m[0,j] = j$
for $i \leftarrow 1$ **to** $|s_1|$
do for $j \leftarrow 1$ **to** $|s_2|$
  do $m[i,j] = \min\{m[i-1,j-1] + \text{if } (s_1[i] = s_2[j]) \text{ then } 0 \text{ else } 1\text{fi},$
    $m[i-1,j] + 1,$
    $m[i,j-1] + 1\}$
return $m[|s_1|, |s_2|]$

Example from Figure 3.5, Introduction to IR, C.D. Manning, P. Raghavan and H. Schütze.

Weighted Edit Distance

- Give different weights based on the operations as well as the characters involved.
- Meant to capture keyboard errors. E.g., **m** is more likely to be mistyped as **n**, rather than **q**.

Edit Distance for Spelling Correction

- **Expensive !**
  - Need to measure the query term distance with every term in the vocabulary.
  - Can we reduce the candidate of potential correct spellings?

Edit Distance for Spelling Correction

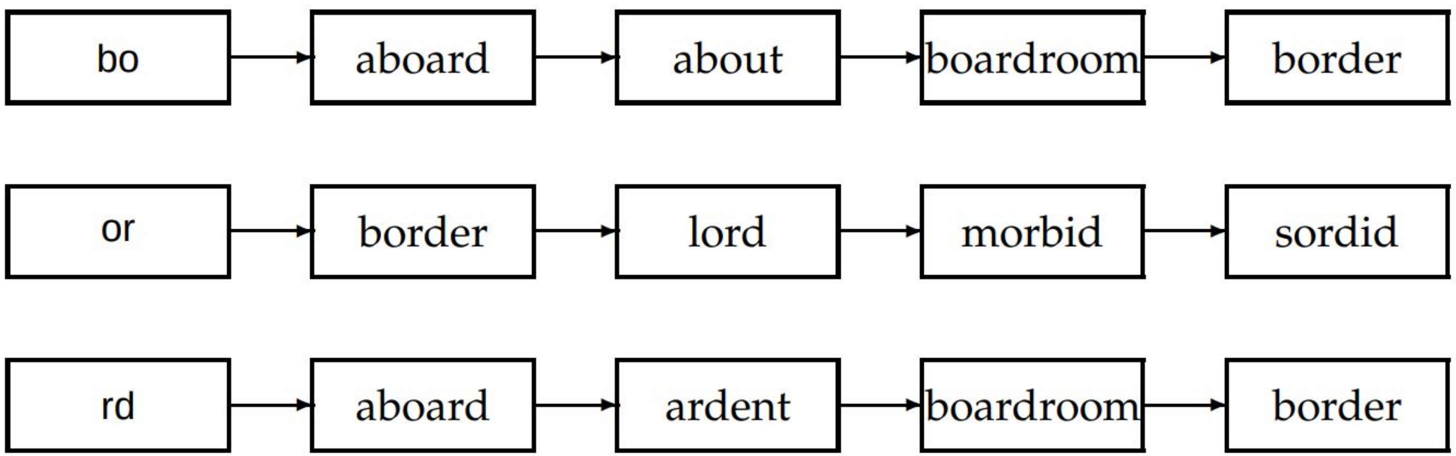
- **Expensive !**
  - Need to measure the query term distance with every term in the vocabulary.
  - Can we reduce the candidate of potential correct spellings?
  - Example heuristic: Only match query term with vocabulary term that starts with same character.

K-gram distance for spelling correction

-

K-gram distance for spelling correction

- Query term: **bord**
- 2-grams: **bo**, **or**, **rd**
- Terms with same 2-grams:



K-gram distance for spelling correction

- Query term: **bord**
- 2-grams: **bo**, **or**, **rd**
- Terms with same 2-grams:

bo → aboard → about → boardroom → border

or → border → lord → morbid → sordid

rd → aboard → ardent → boardroom → border

- Shall **boardroom** be a plausible correct spelling?

K-gram distance for spelling correction

- While matching k-grams, we can use heuristics such as:
  - At-least **n** number of k-grams should match.
  - The query term and correct spelling can only differ by at most **n** k-grams.
  - Set overlap using Jaccard coefficient:
    - $|A \cap B| \,/\, |A \cup B|$

Context Specific Spelling Correction

Context Specific Spelling Correction

- Query: “flew <u>form</u> Delhi”
- Query: “... fifteen minuets ... ”

Context Specific Spelling Correction

- Query: “flew <u>form</u> Delhi”
- Simple idea (**hit-based**): Enumerate correction for each of the query term.
  - flew → flow, flea
  - form → from
- Try different combinations:
  - flea form Delhi
  - flew from Delhi
  - flow from Delhi

Context Specific Spelling Correction

- Query: "flew form Delhi"
- Simple idea (**hit-based**): Enumerate correction for each of the query term.
  - flew → flow, flea
  - form → from
- Try different combinations:
  - flea form Delhi
  - flew from Delhi
  - flow from Delhi
- The query that has most hits, will be considered as correct.

Context Specific Spelling Correction

- Query: “flew <u>form</u> Delhi”
- Simple idea (**hit-based**): Enumerate correction for each of the query term.
    - flew → flow, flea
    - form → from
- Try different combinations:
    - flea form Delhi
    - flew from Delhi
    - flow from Delhi
- The query that has most hits, will be considered as correct.
- More efficient alternatives looks 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.

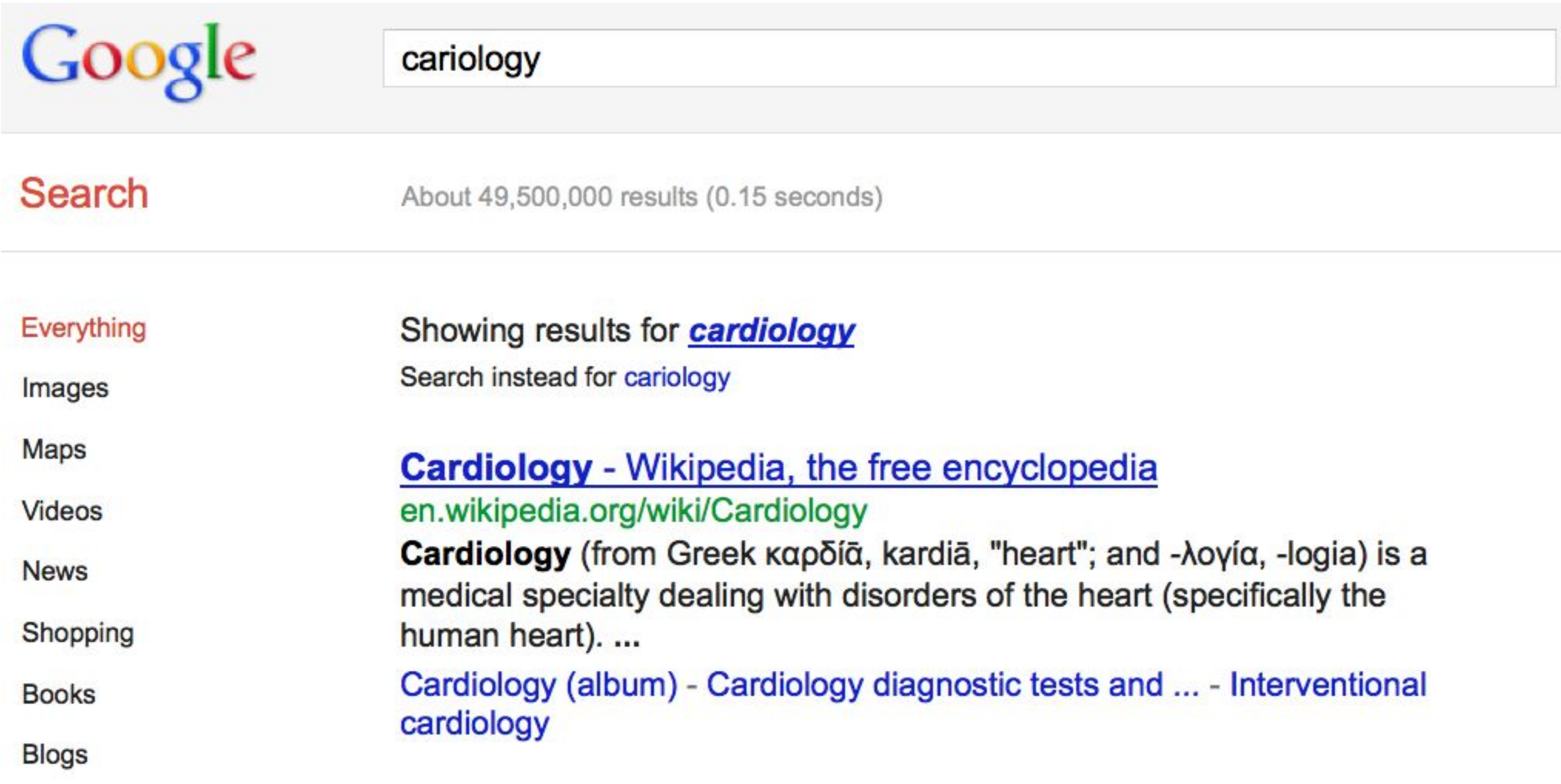
Spelling Correction on Documents???

- In Information Retrieval the general philosophy is to not change documents.
- However, if the documents are created by using Optical Character Recognition (OCR) techniques, then spelling correction on documents can be performed.

Correct Word Sources

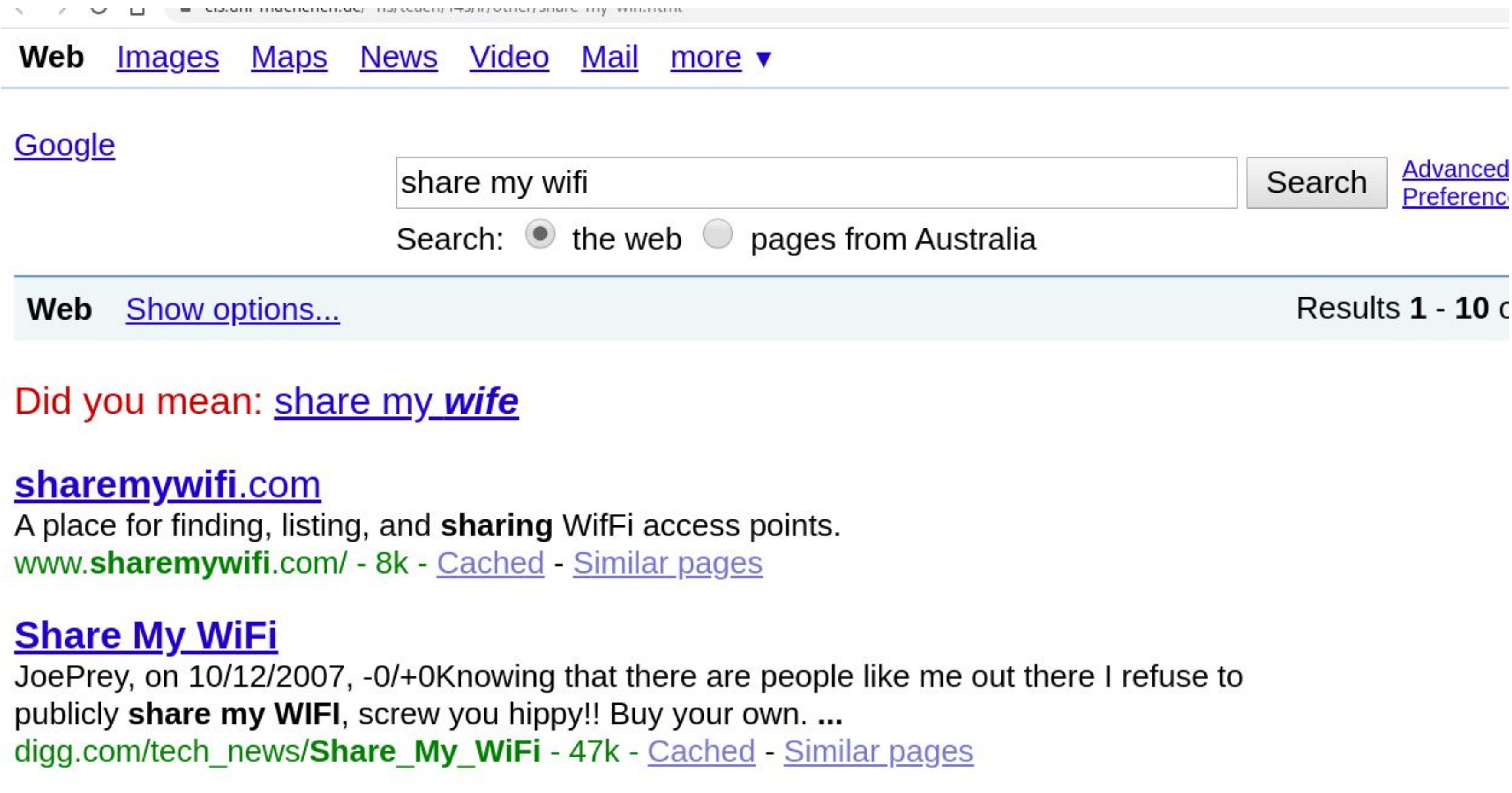
- All the words appears in the document collection?
- Standard dictionaries (Webster's, Oxford, …)?
- Specific dictionary for specialized IR systems?

Spelling Correction Gone Wrong



Cariology: The study of dental caries and their development.

Spelling Correction Gone Wrong



Reference

https://nlp.stanford.edu/IR-book/ (Chapter 3)

Lecture slides on Dictionaries & tolerant retrieval:

https://www.cis.uni-muenchen.de/~hs/teach/14s/ir/pdf/03dict.flat.pdf

Assignment 1

- ~~Marks 10~~ Weightage: 10%
- Last date of submission: 7th Feb 2020
- To be submitted Individually
- There will be code and a report in the submission.
- Detailed submission guidelines will be posted later.

Next Lecture

- **Index Construction**
  - The data size is enormous for several real world IR systems.
  - Example:
    - Over billion web pages on the Internet!
    - IR systems also maintain history of web pages.
  - Datasets:
    - **http://commoncrawl.org/**
    - 25 billion web pages
    - Petabytes of data

Thank You!