Introduction to

Information Retrieval

03 Dictionaries and tolerant retrieval

3.1 Search structures for dictionaries
3.2 Wildcard queries
3.3 Spelling correction

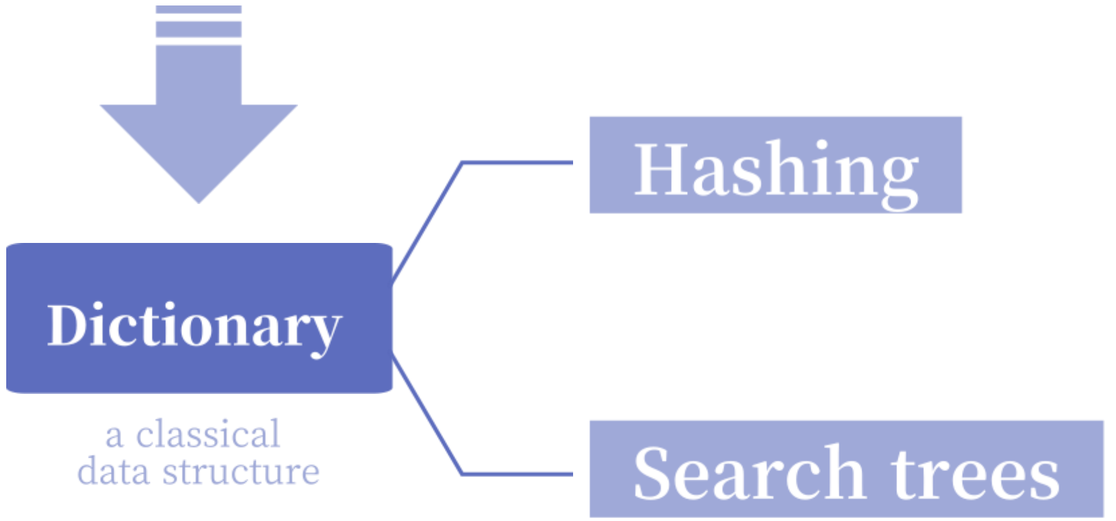
BK인턴 유승미

Contents

Search structures for dictionaries

- Given an inverted index and a query

  1 to determine whether each query term exists in the vocabulary



Search structures for dictionaries

Hashing

Each vocabulary term (key) is hashed into an integer over a large enough space that hash collisions are unlikely.

Issues

- Possibility of hash collisions.
- There is no easy way to find minor variants of a query term, since these could be hashed to very different integers.
  (such as the accented and non-accented versions of a word like resume)
- In a setting (such as the Web) where the size of the vocabulary keeps growing, a hash function designed for current needs may not suffice in a few years' time.

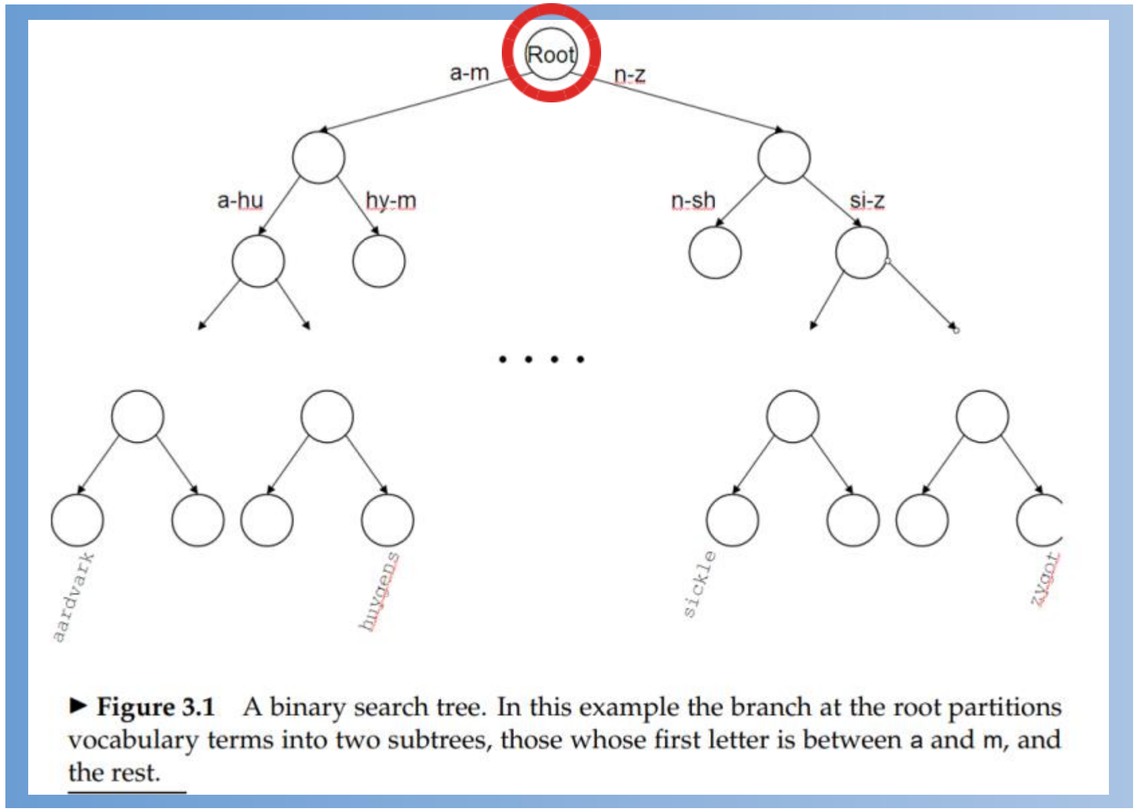
Search structures for dictionaries

Search trees ← overcome many of hashing's issues

best-known

Binary tree

each internal node has two children



► **Figure 3.1** A binary search tree. In this example the branch at the root partitions vocabulary terms into two subtrees, those whose first letter is between a and m, and the rest.

Search structures for dictionaries

Search trees ← overcome many of hashing's issues

best-known

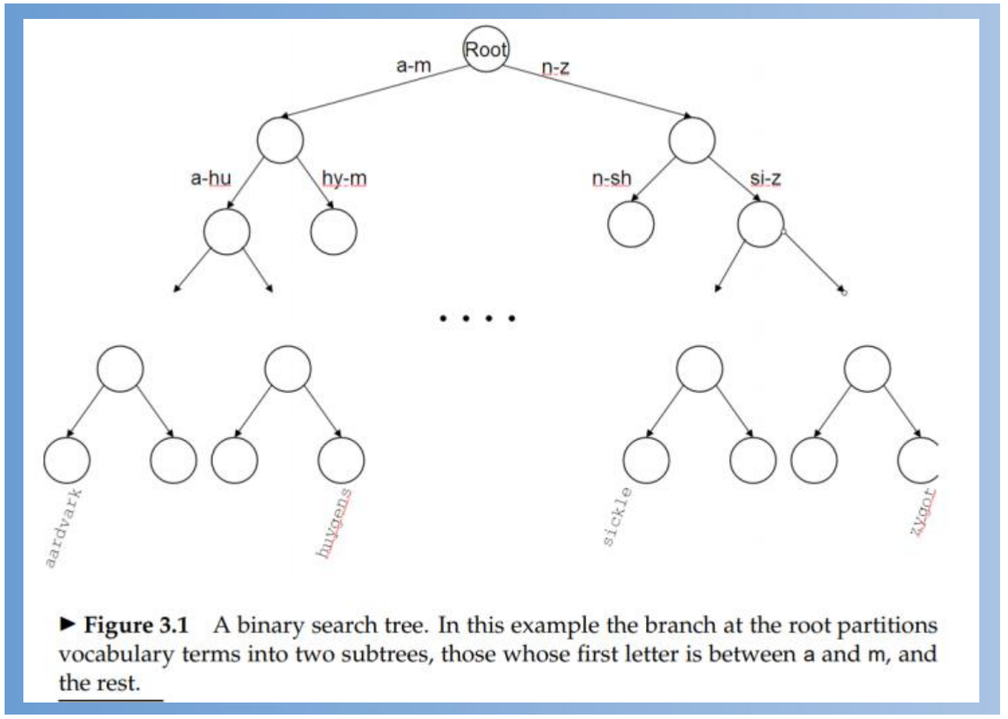
Binary tree

- for instance, they permit us to enumerate all vocabulary terms beginning with *automat*.
- Efficient search hinges on the tree being balanced.

The principal issue

Rebalancing

as terms are inserted into or deleted from the binary search tree, it needs to be rebalanced so that the balance property is maintained.



► **Figure 3.1** A binary search tree. In this example the branch at the root partitions vocabulary terms into two subtrees, those whose first letter is between a and m, and the rest.

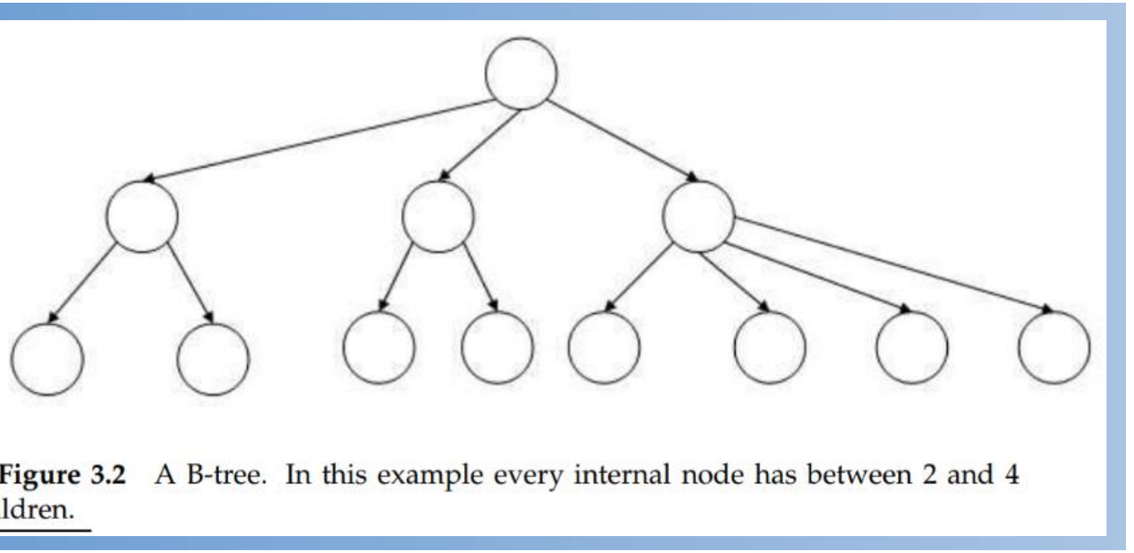
Search structures for dictionaries

B-tree commonly used for a dictionary

a search tree in which every internal node has a number of children in the interval [a, b]

**a,b: appropriate positive integers*

- A B-tree may be viewed as "collapsing" multiple levels of the binary tree into one.



► **Figure 3.2** A B-tree. In this example every internal node has between 2 and 4 children.

Wildcard queries

- Wildcard queries are used in any of the following situations:

1. the user is uncertain of the spelling of a query term.
   (e.g., Sydney vs. Sidney, which leads to the wildcard query S*dney)
2. the user is aware of multiple variants of spelling a term
   and (consciously) seeks documents containing any of the variants.
   (e.g., color vs. colour)
3. the user seeks documents containing variants of a term that would be caught by stemming, but is unsure whether the search engine performs stemming.
   (e.g., judicial vs. judiciary, leading to the wildcard query judicia*)
4. the user is uncertain of the correct rendition of a foreign word or phrase.
   (e.g., the query Universit* Stuttgart)

Wildcard queries

- Type of wildcard query

Wildcard queries

Trailing wildcard query

e.g.,

mon* → **"B-Tree"**

1. Walk down the tree following the symbols of the prefix (e.g., "m", "o", "n" for "mon").
2. Enumerate the set W of terms in the dictionary with the prefix "mon".
3. Perform $|W|$ lookups on the standard inverted index to retrieve all documents containing any term in W.

Wildcard queries

Leading wildcard query

e.g.,

mon** → **"reverse B-Tree"

1. Each root-to-leaf path in the reverse B-tree corresponds to a term written backwards.

2. For a given prefix, walk down the reverse B-tree to enumerate all terms R in the vocabulary with that prefix

e.g.,

lemon** → **root-n-o-m-e-l

Wildcard queries

- We can handle an even more general case:

e.g.,

to enumerate the set W of dictionary terms beginning with the prefix *se*

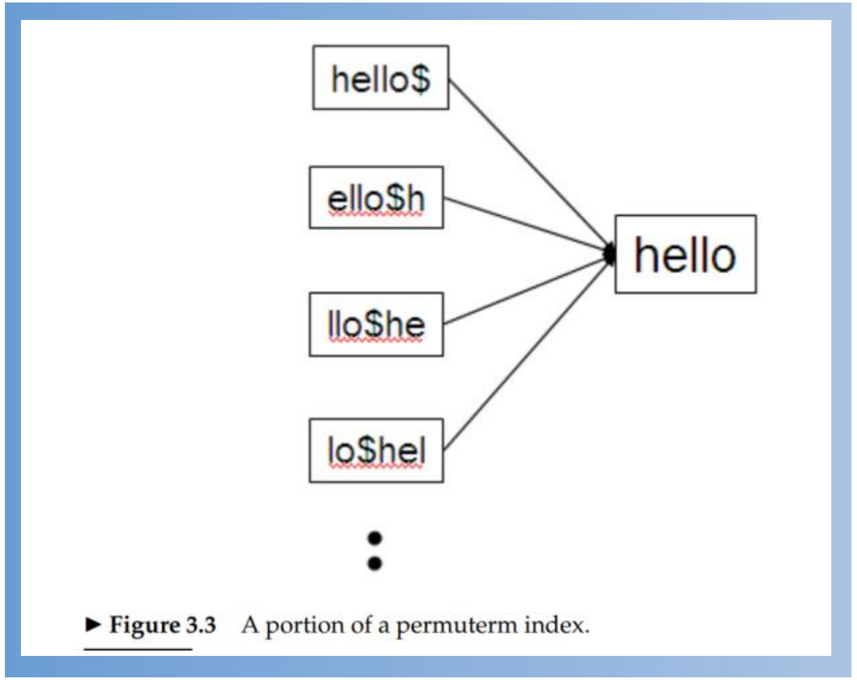
enumerate the set R of terms ending with the suffix *mon*

take the intersection $W \cap R$ of these two sets

General wildcard queries

1 Permuterm indexes (a form of *inverted index*)

a permuterm index, in which the various rotations of each term (augmented with $) all link to the original vocabulary term.



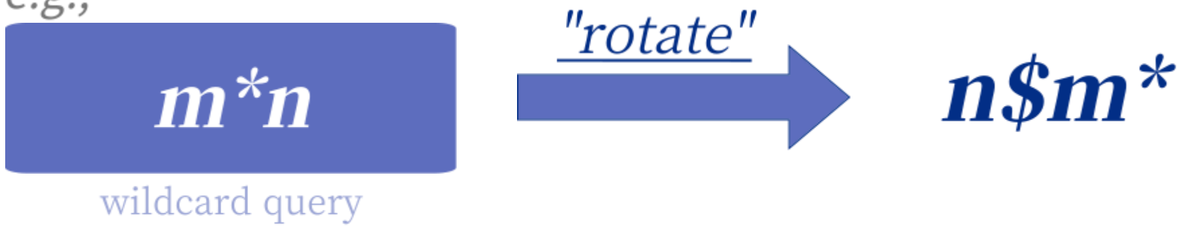
► **Figure 3.3** A portion of a permuterm index.

*a special symbol $: to mark the end of a term

General wildcard queries

1 Permuterm indexes

e.g.,



- Next, we look up this string in the permuterm index, where seeking *n$m**
  (via a search tree)

the permuterm index enables us to identify the original vocabulary terms matching a wildcard query

General wildcard queries

1 Permuterm indexes

disadvantage

Its dictionary becomes quite large, including as it does all rotations of each term.

e.g.,

fi*mo*er → ***er$fi**** + ***filter out terms that do not contain "mo"***

fishmonger (O)
filibuster (X)

General wildcard queries

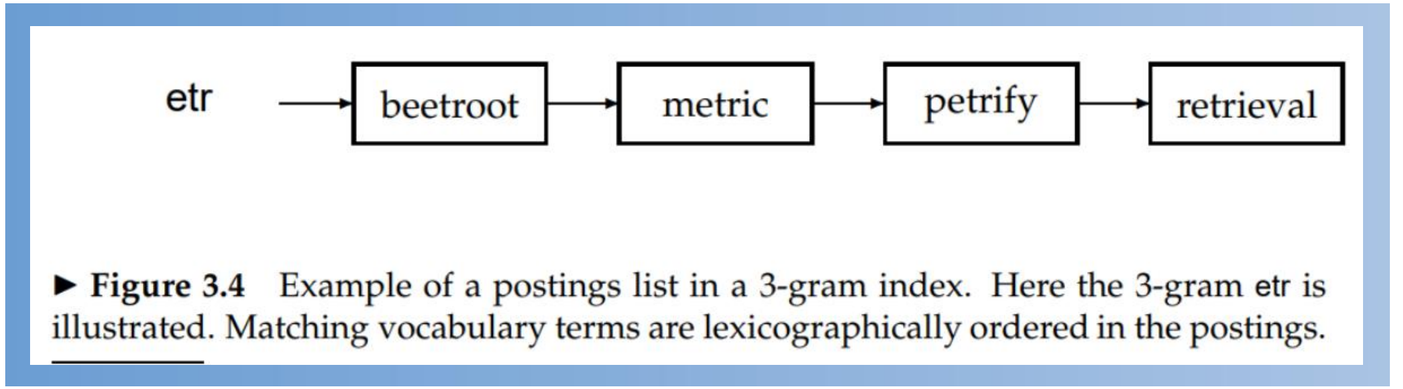
2 k-gram indexes

: a sequence of *k* characters

e.g.,

castle —*3-grams*→ ***$ca cas ast stl tle le$***

- Each postings list points from a k-gram to all vocabulary terms containing that k-gram.



► **Figure 3.4** Example of a postings list in a 3-gram index. Here the 3-gram etr is illustrated. Matching vocabulary terms are lexicographically ordered in the postings.

General wildcard queries

2 k-gram indexes

e.g.,

re*ve

wildcard query

looked up in the standard inverted index
to yield documents matching the query

Boolean query

$re AND ve$

looked up in the 3-gram index

matched

relive

remove

retrieve

General wildcard queries

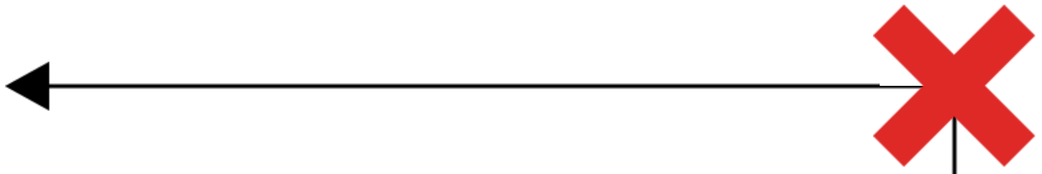
2 k-gram indexes

→ demands one further step of processing.

e.g.,

red*

wildcard query



< solution >

"post-filtering step"

check individually
against the original query

Boolean query

$re* AND *red

matched

retired

looked up in the 3-gram index

Spelling correction

- Two steps to solving *spelling correction*

1. Edit distance
2. K-gram overlap

Spelling correction

1 Edit distance → *sometimes known as "Levenshtein distance"*

- Given two character strings s1 and s2,

  the "edit distance" between them is the minimum number of edit operations required to transform s1 into s2.

Edit operations

(i) insert a character into a string;

(ii) delete a character from a string;

(iii) replace a character of a string by another character;

Spelling correction

1 Edit distance

```
EDITDISTANCE(s1, s2)
 1  int m[i, j] = 0
 2  for i ← 1 to |s1|
 3  do m[i, 0] = i
 4  for j ← 1 to |s2|
 5  do m[0, j] = j
 6  for i ← 1 to |s1|
 7  do for j ← 1 to |s2|
 8     do m[i, j] = min{m[i − 1, j − 1] + if (s1[i] = s2[j]) then 0 else 1fi,
 9                      m[i − 1, j] + 1,
10                      m[i, j − 1] + 1}
11  return m[|s1|, |s2|]
```

substituting a character in s1 ↑

→ inserting a character in s1

→ inserting a character in s2

► **Figure 3.5** Dynamic programming algorithm for computing the edit distance between strings s_1 and s_2.

the min of the other three

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

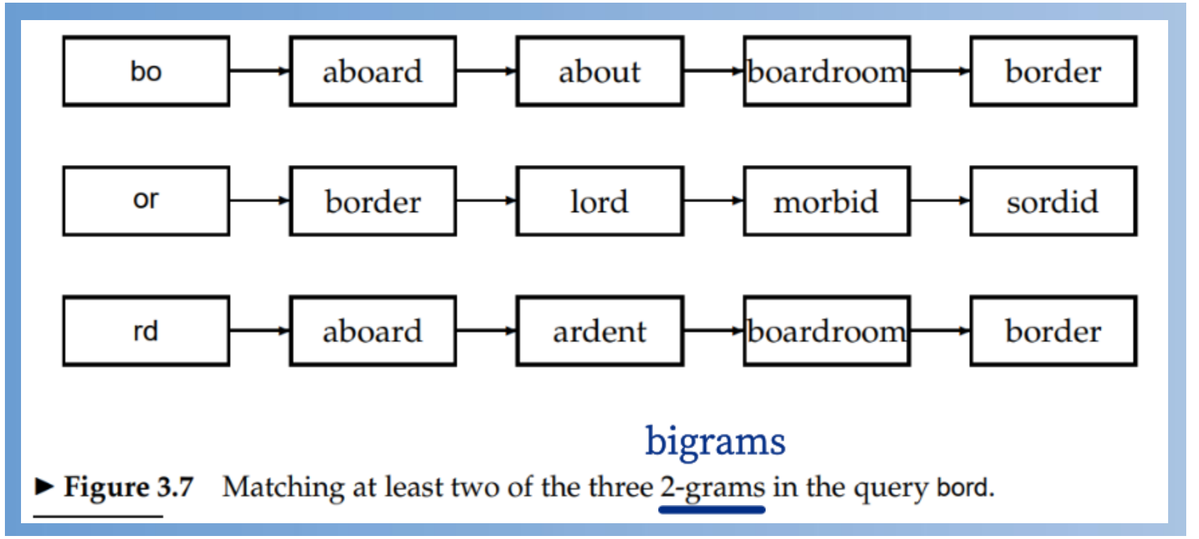
► **Figure 3.6** Example Levenshtein distance computation. The 2×2 cell in the $[i, j]$ entry of the table shows the three numbers whose minimum yields the fourth. The cells in italics determine the edit distance in this example.

Spelling correction

2 K-gram indexes

- To further limit the set of vocabulary terms for which we compute edit distances to the query term, we now show how to invoke the **k-gram index** to assist with retrieving vocabulary terms with low edit distance to the *query q*.

e.g.,



► **Figure 3.7** Matching at least two of the three 2-grams in the query bord.

aboard

boardroom

border

implausible "correction"

Spelling correction

2 K-gram indexes

- We require more nuanced measures of the overlap in k-grams between a vocabulary term and q.

A

the set of k-grams in the *query q*

B

the set of k-grams in a *vocabulary term*

Jaccard coefficient

$$\frac{|A \cap B|}{|A \cup B|}$$

e.g.,
- query: bord
- term: boardroom

$$\frac{\underline{2}}{(\underline{8} + \underline{3} - 2)}$$

bo / rd

bo / or / ar / rd / dr / ro / oo / om

bo / or / rd

감사합니다.