21.11.2012

Zaglądamy pod maskę: podstawy działania silnika wyszukiwawczego na przykładzie Lucene

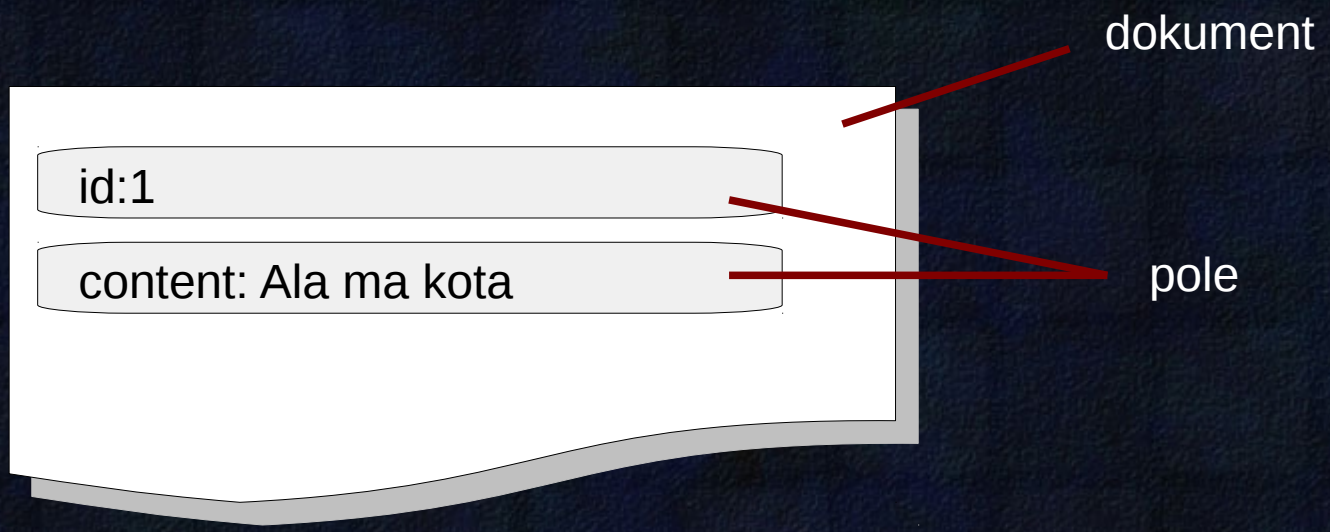
Dominika Puzio

Indeks

Podstawy: dokument

Dokument:

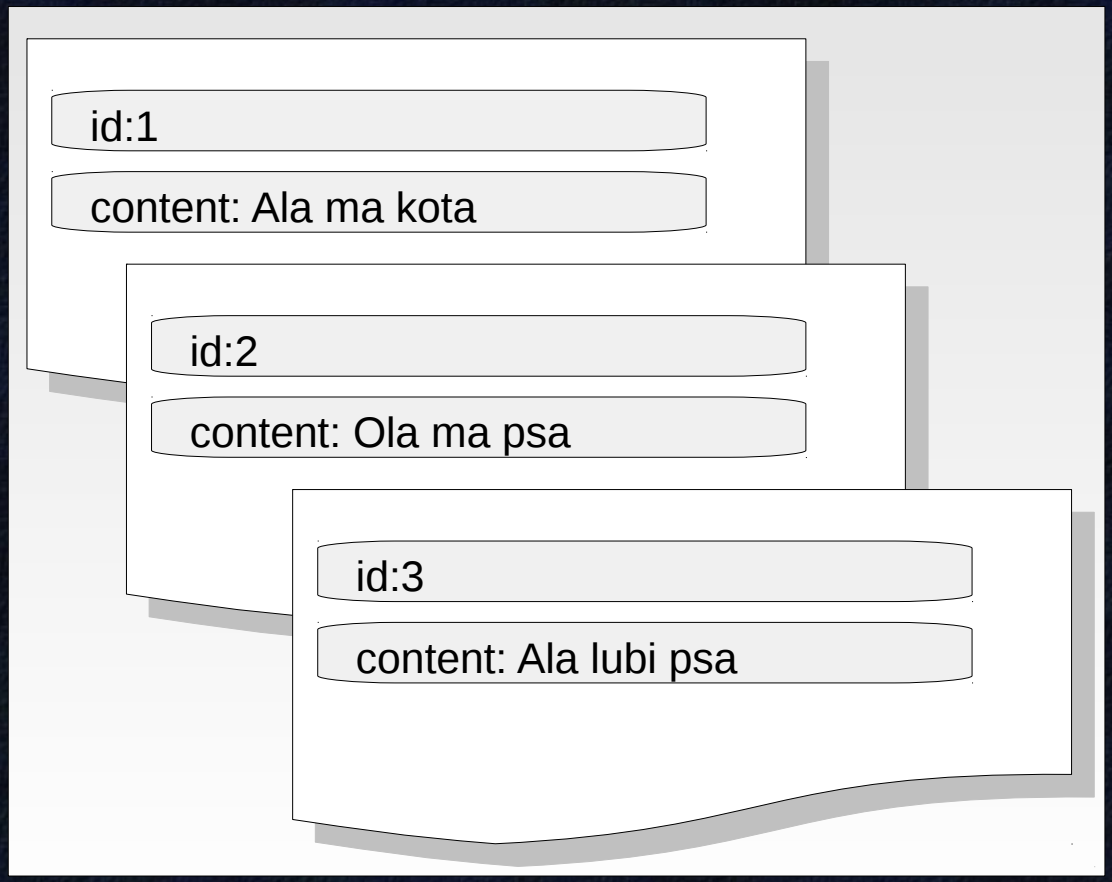
jednostka danych, pojedynczy element na liście wyników wyszukiwania, to co chcemy wyszukiwać (strona www, artykuł, post, dobro konsumpcyjne)



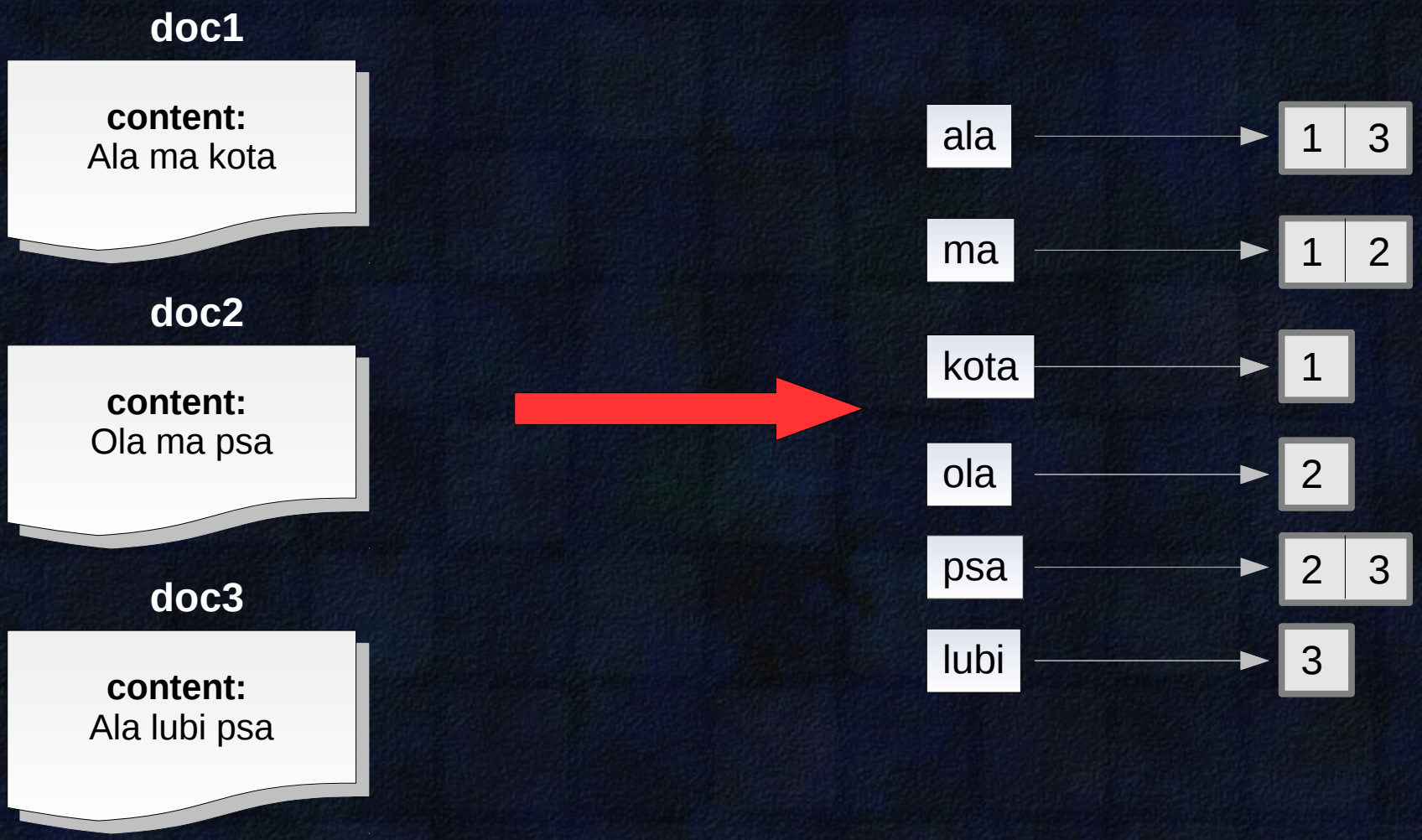
Podstawy: indeks

Indeks:

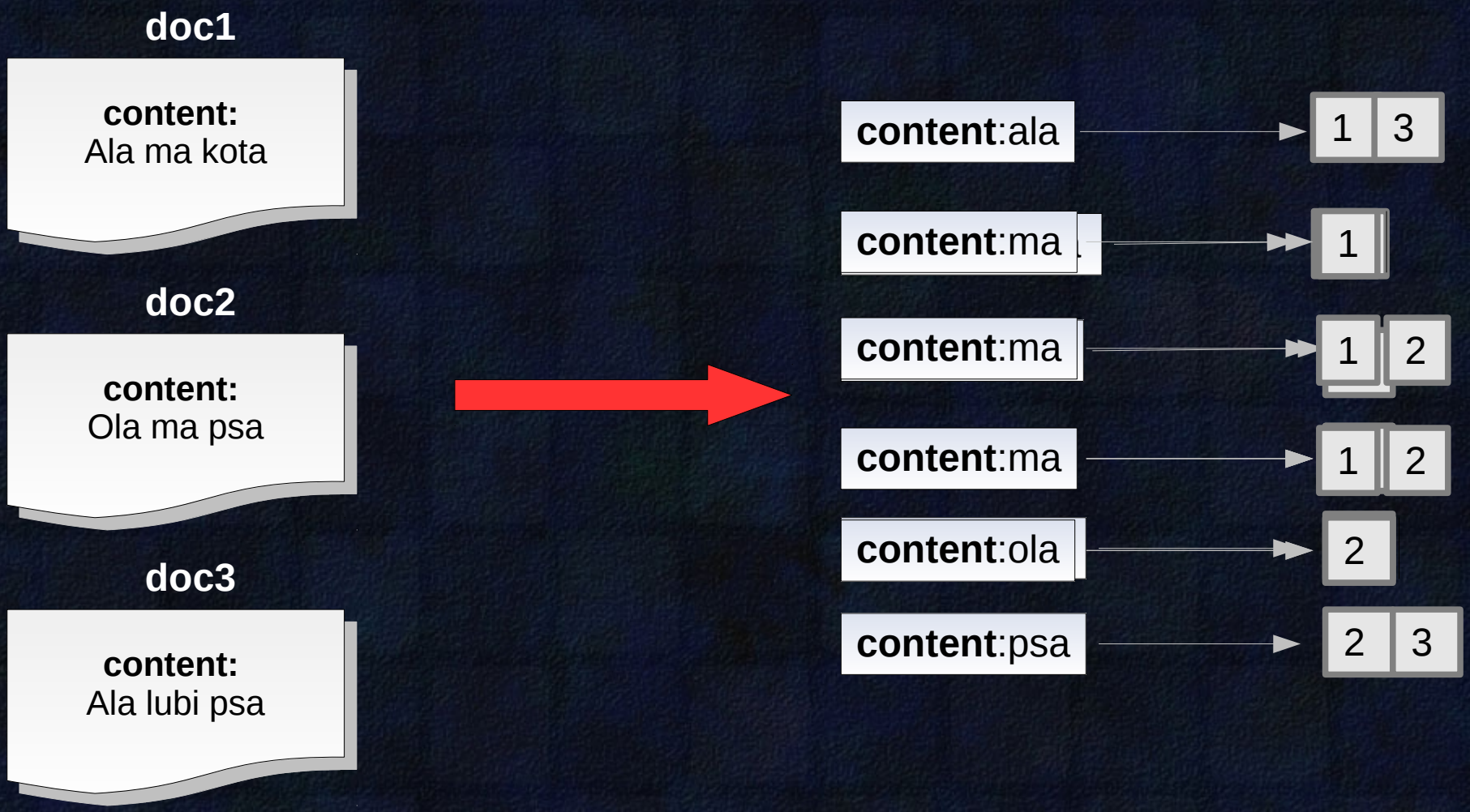
repozytorium, w którym silnik wyszukiwawczy przechowuje dane, zorganizowane w taki sposób, aby dało się w nich szybko wyszukiwać



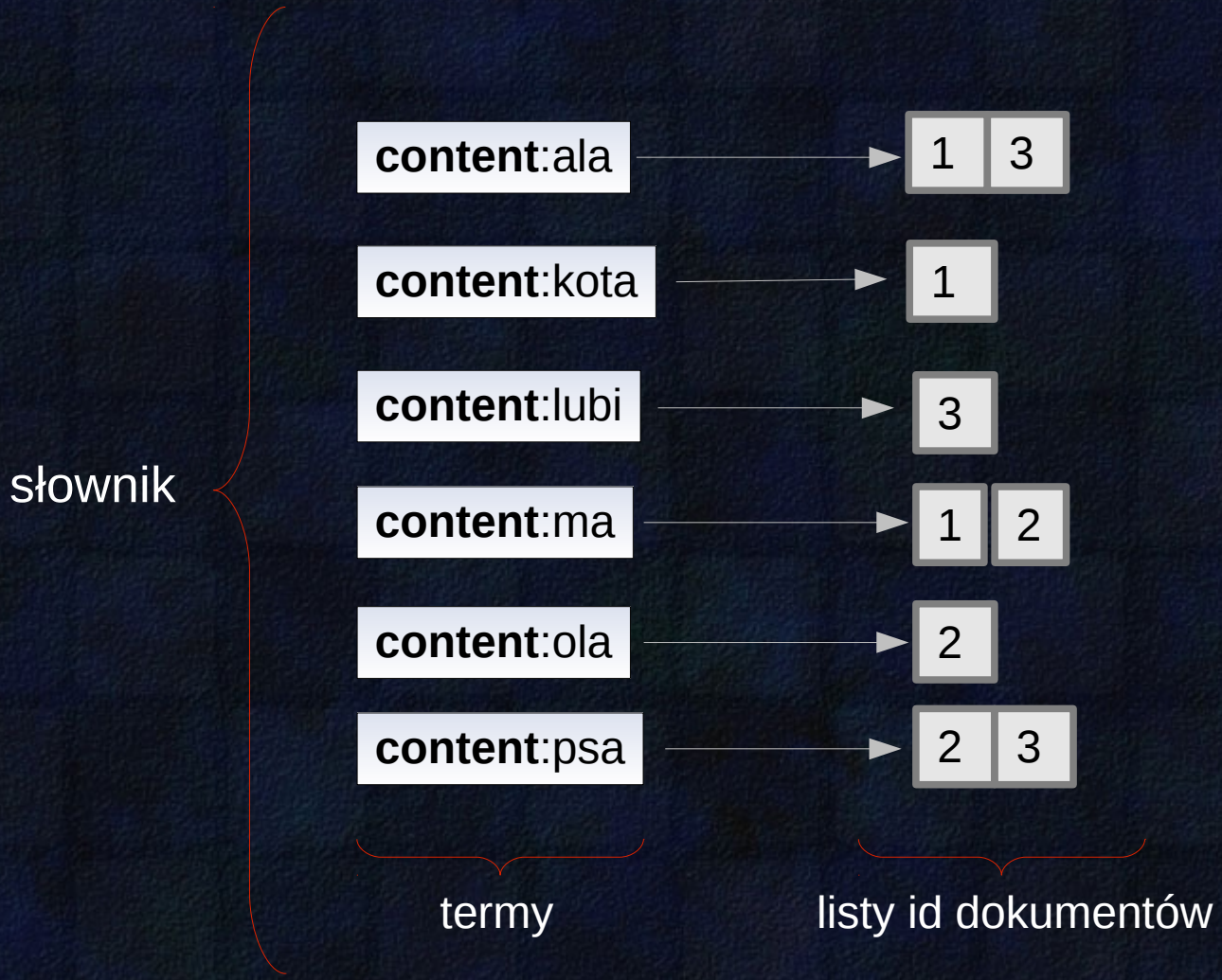
Indeks odwrócony



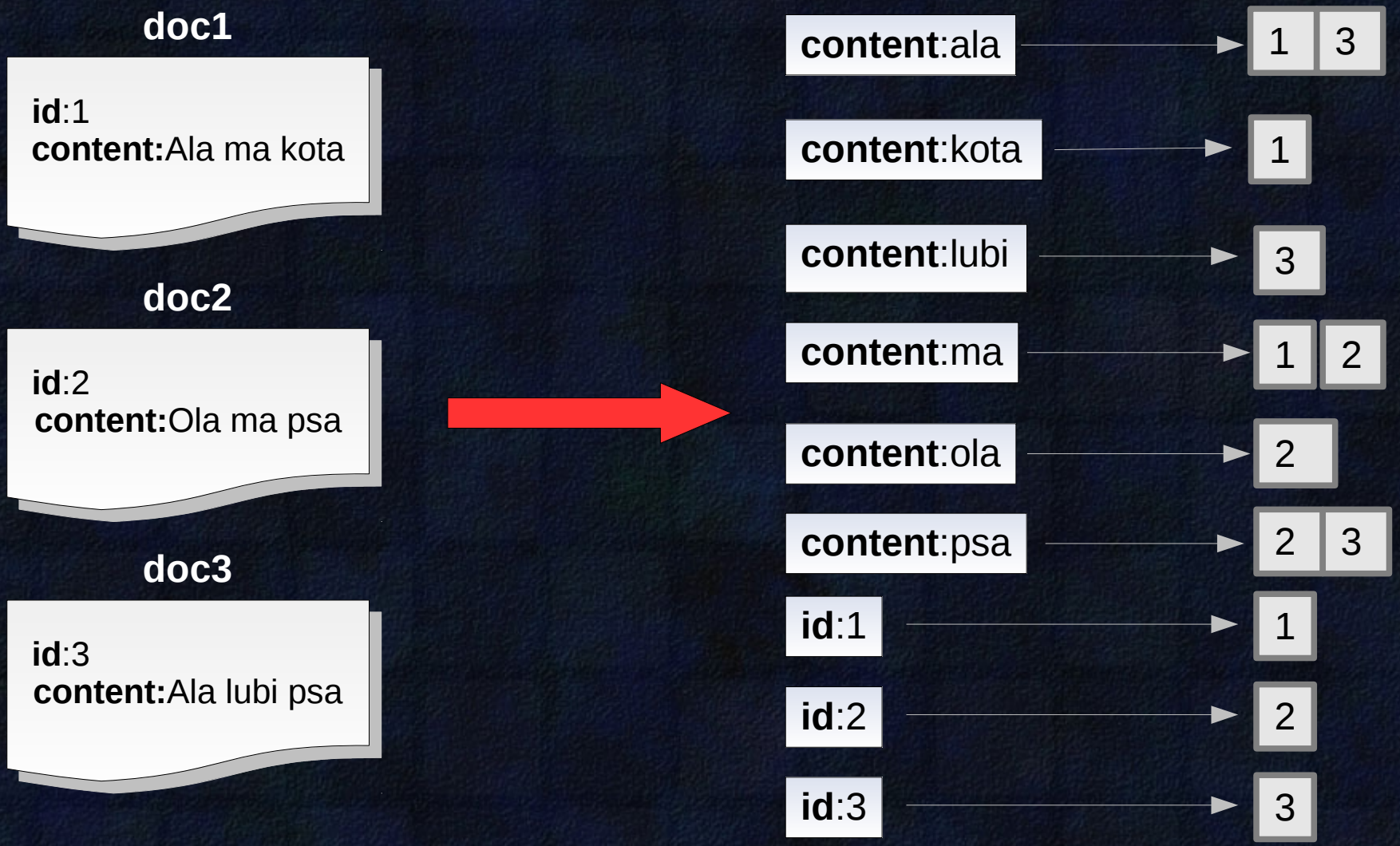
Indeks: budowanie



Indeks: słownik



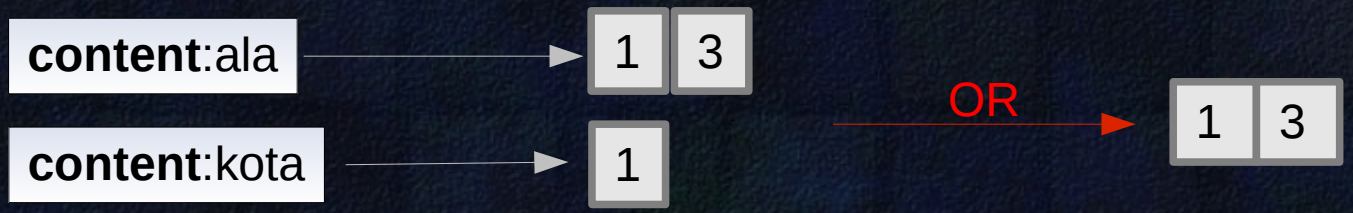
Indeks: słownik



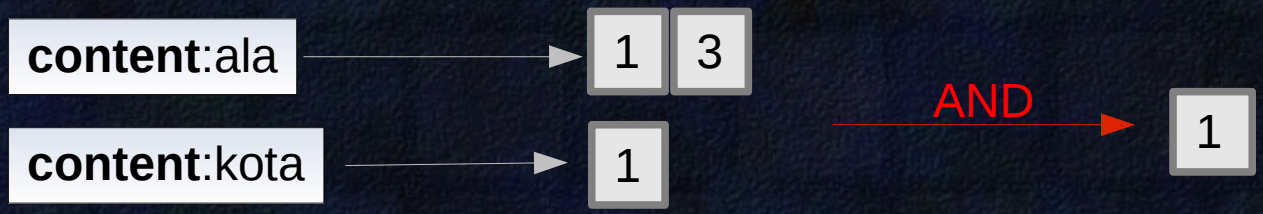
Indeks: przebieg wyszukiwania

- QUERY: **[**content: ala**]**

- QUERY: **[**content: ala **OR** content: kota**]**



- QUERY: **[**content: ala **AND** content: kota**]**



Indeks: oglądamy słownik

Luke – czytnik indeksów Lucene

Algorytm scoringu

(czyli dlaczego kot jest ważniejszy niż pies)

Vector Space Model i trochę historii

- **historycznie:**
  - idea użycia komputerów do poszukiwania informacji pojawiła się w 1945 roku (http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/)
  - pierwsze silniki wyszukiwawcze (akademickie) – lata 50'
  - pierwsze wielkoskalowe silniki wyszukiwawcze, gotowe do użytku komercyjnego – lata 70'

- **Vector Space Model:** algebraiczny model dokumentu tekstowego, opracowany i pierwszy raz użyty w latach 60' na Cornell University jako część systemu SMART (System for the Mechanical Analysis and Retrieval of Text) – jednej z pierwszych tekstowych wyszukiwarek
- **Vector Space Ranking:** algorytm scoringu oparty o *Vector Space Model*

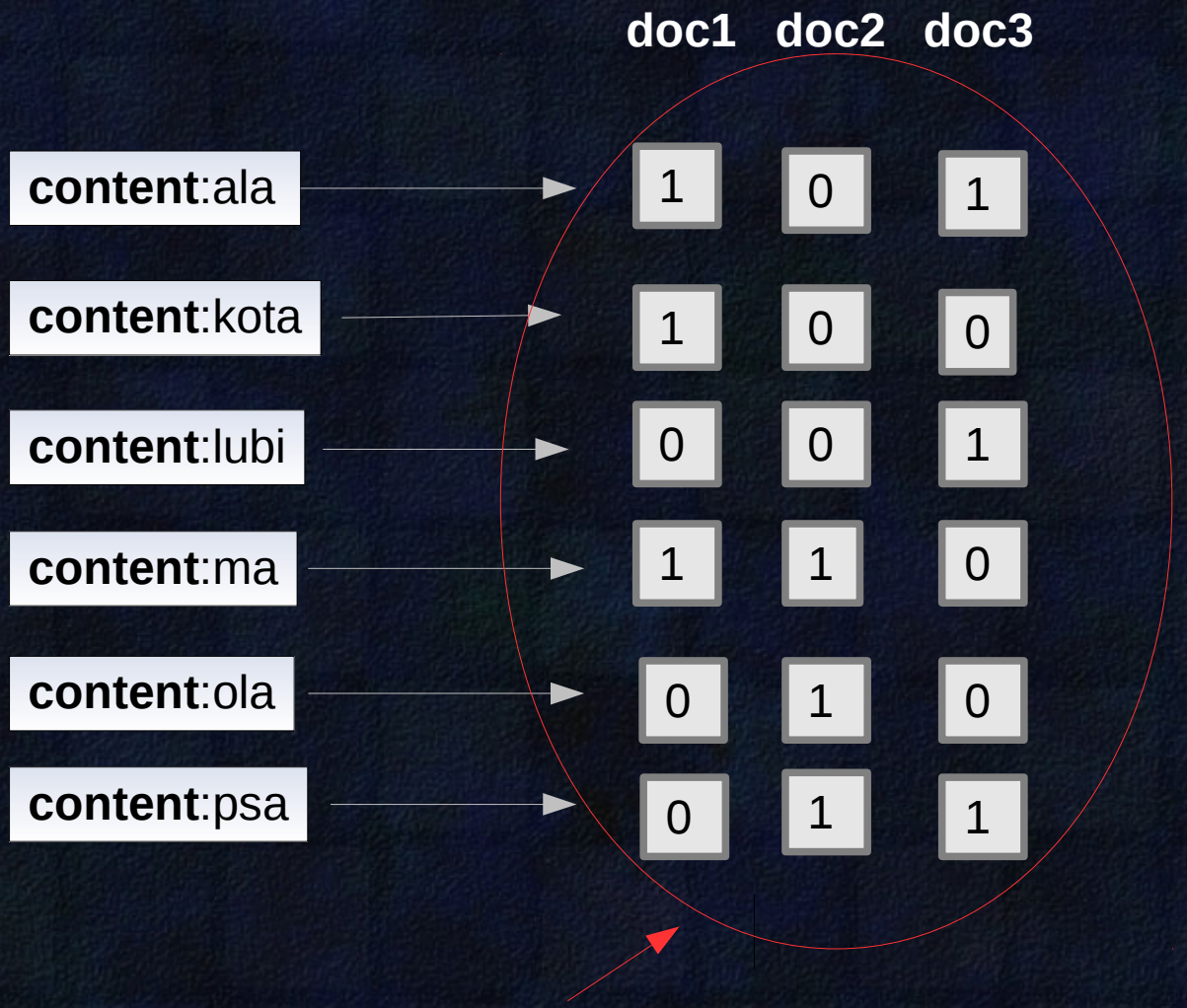
Informacje zawarte w słowniku

dokument 1: zawiera słowo „ala”

dokument 2: nie zawiera słowa „ala”

dokument 3: zawiera słowo „ala”

Alternatywny zapis słownika



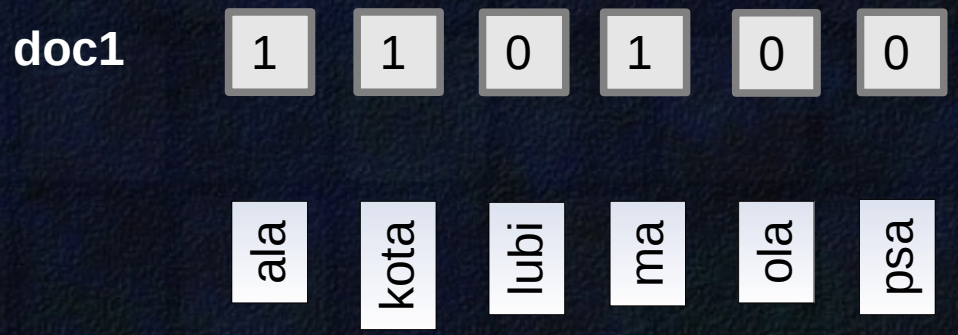
Vector Space Model: wektor dokumentu

	doc1	doc2	doc3
ala	1	0	1
kota	1	0	0
lubi	0	0	1
ma	1	1	0
ola	0	1	0
psa	0	1	1

	ala	kota	lubi	ma	ola	psa
doc3	1	0	1	0	0	1
doc2	0	0	0	1	1	1
doc1	1	1	0	1	0	0

wektor dokumentu doc1 w przestrzeni słów

Vector Space Model: waga słów



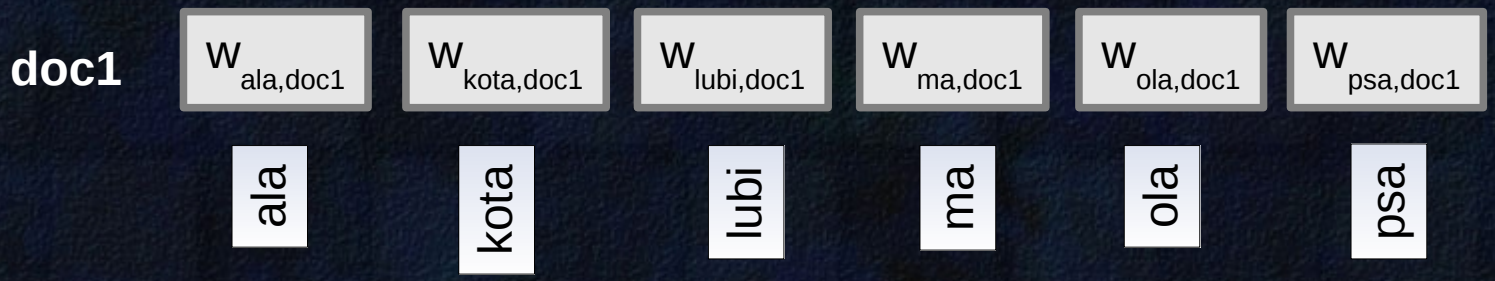
Założenie Vector Space Model: informacja o tym, że słowo wystąpiło w dokumencie (lub nie) nie jest wystarczająca.

Model bierze pod uwagę również to, że:

- im więcej wystąpień słowa w dokumencie, tym wyżej powinien być na liście wyników
- nie wszystkie słowa są jednakowo ważne

Vector Space Model: waga słów

Wartością współrzędnej słowa *s* w dokumencie *d* jest waga $w_{s,d}$ obliczana na podstawie liczby wystąpień słowa *s* w dokumencie *d* i wartości informacyjnej słowa *s*.



Vector Space Model

Krok 1: każde słowo w dokumencie dostaje wagę zależną od liczby wystąpień słowa w dokumencie i wartości informacyjnej słowa

- bierzemy pod uwagę liczbę wystąpień poszukiwanego słowa w dokumencie - im więcej razy słowo z zapytania wystąpiło w dokumencie, tym wyższy *score* dla dokumentu
- dla każdego słowa w dokumencie mamy liczbę:

  $tf_{s,d}$ *(term frequency)* = liczba wystąpień słowa s w dokumencie *d*

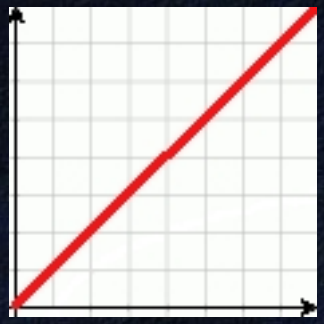
- dla każdego słowa w dokumencie wyznaczamy wagę wg. wzoru:

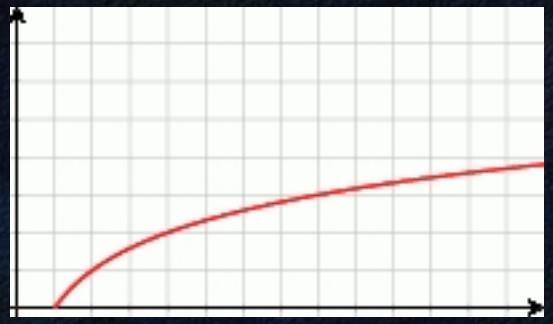
$$w(s,d) = \begin{cases} 1+\log(tf_{s,d}) & dla\ tf_{s,d}>0 \\ 0 & w\ przeciwnym\ wypadku \end{cases}$$

Vector Space Model

$$w(s,d) = \begin{cases} 1+\log(tf_{s,d}) & dla\ tf_{s,d} > 0 \\ 0 & w\ przeciwnym\ wypadku \end{cases}$$

- Dlaczego logarytm?
  - żeby nie mieć zależności wprost (dokument zawierający słowo 10 razy, nie jest 10 razy lepszy od tego, który zawiera je raz)





- Dlaczego *1+log ?*
  - żeby słowo, które wystąpiło w dokumencie 1 raz, nie otrzymało wagi 0 (log1 = 0)
- Dlaczego osobno przypadek, kiedy $tf_{s,d} = 0$?
  - dlatego, że *log0 = -∞*

Vector Space Model

Krok 1: każde słowo w dokumencie dostaje wagę zależną od liczby wystąpień słowa w dokumencie i wartości informacyjnej słowa

- bierzemy pod uwagę liczbę wystąpień poszukiwanego słowa w całym indeksie – im rzadsze słowo, tym więcej informacji niesie i dokument je zawierający powinien mieć wyższy score
- dla każdego słowa w dokumencie mamy liczbę:

df_s *(document frequency)* = liczba dokumentów w indeksie zawierających słowo *s*

- dla każdego słowa w dokumencie wyznaczamy tzw. *inverse document frrequency:*

$$idf_s = \log\left(\frac{N}{df_s}\right)$$

N - liczba wszystkich dokumentów w indeksie

Vector Space Model

Krok 1: każde słowo w dokumencie dostaje wagę zależną od *tf* i *idf*

- dla każdego słowa w dokumencie wyznaczamy wagę:

$$w_{s,d} = (1 + \log(tf_{s,d})) \times \log(\frac{N}{df_s})$$

- waga jest wyznaczana podczas indeksowania i zapisywana w indeksie

Przykład:

	doc1	doc2	doc3
ala	1	0	1

$tf_{ala,\,doc1} = tf_{ala,doc3} = 1$

$tf_{ala,doc2} = 0$

$N = 3$

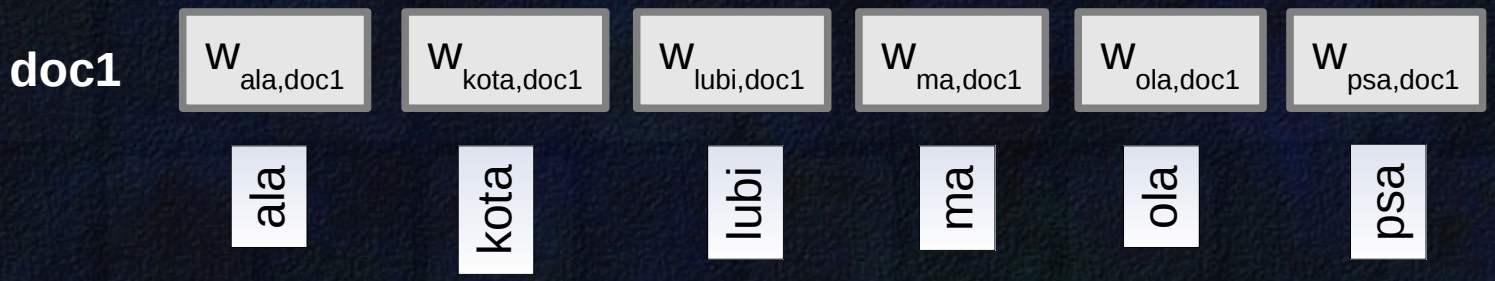
$df_{ala} = 2$

$$w_{ala,doc1} = w_{ala,doc3} = (1 + \log(1)) \times \log(\frac{3}{2}) \approx 0{,}58$$

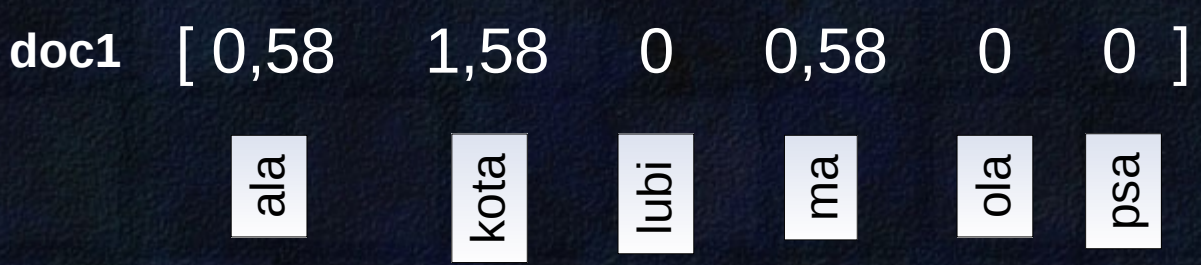
$$w_{ala,doc2} = 0 \times \log(\frac{3}{2}) = 0$$

Vector Space Model

Krok 2: każdy dokument przedstawiamy jako wektor w przestrzeni słów



doc1: Ala ma kota



Vector Space Model

Krok 2: każdy dokument przedstawiamy jako wektor w przestrzeni słów

doc1: Ala ma kota
doc2: Ola ma psa
doc3: Ala lubi psa

	ala	kota	lubi	ma	ola	psa
doc1	0,58	1,58	0	0,58	0	0
doc2	0	0	0	0,58	1,58	0,58
doc3	0,58	0	1,58	0	0	0,58

Vector Space Model (Ranking)

Krok 3: zapytanie przedstawiamy jako wektor w przestrzeni słów

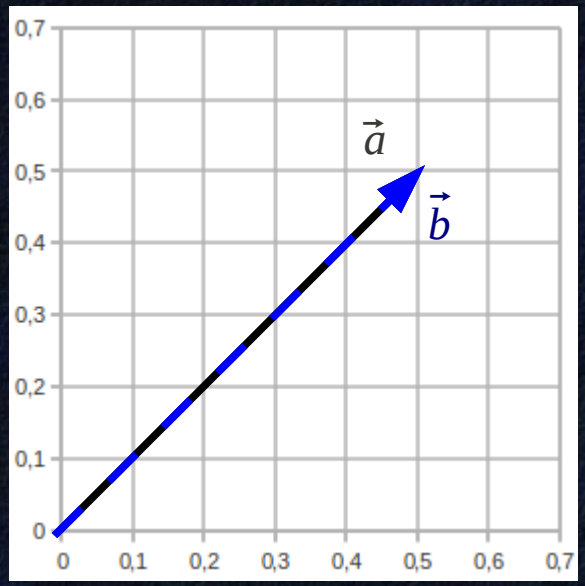
z: kota i psa

z	[	0	1,58	0	0	0	0,58]
		ala	kota	lubi	ma	ola	psa

Vector Space Model (Ranking)

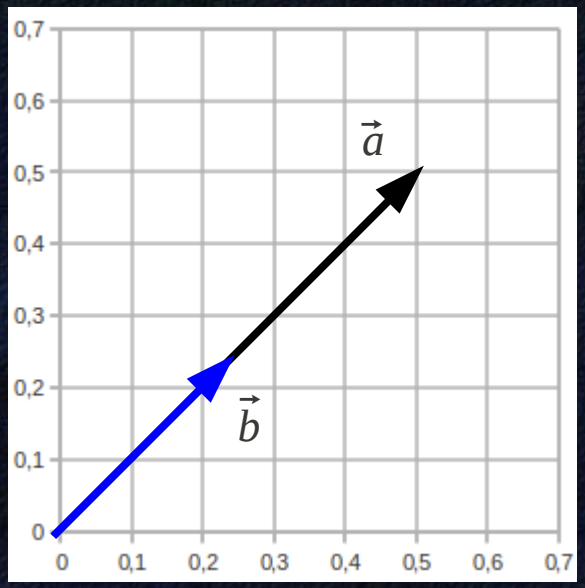
Krok 4: liczymy odległość pomiędzy wektorem zapytania a wektorem dokumentu

- odległość Euklidesowa nie jest dobra:



$$d(\vec{a},\vec{b})=0$$

$$\sphericalangle(\vec{a},\vec{b})=0$$



$$d(\vec{a},\vec{b})=\sqrt{(0.5-0.25)^2+(0.5-0.25)^2}\approx 0.35$$

$$\sphericalangle(\vec{a},\vec{b})=0$$

Vector Space Model (Ranking)

Krok 4: liczymy odległość pomiędzy wektorem zapytania a wektorem dokumentu

- jako miary odległości można użyć kąta między wektorami – ale obliczanie kąta jest trudne (π !)
- zamiast samego kąta można policzyć jego *cosinus:*

$$\cos(\sphericalangle(\vec{a},\vec{b})) = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$$

- *cos0˚ = 1*
- *cos90˚ = 0*

***cosinus* kąta między wektorami = odległość cosinusowa wektorów**

Vector Space Model (Ranking)

1. każde słowo w dokumencie dostaje wagę zależną od liczby wystąpień słowa w dokumencie i wartości informacyjnej słowa
2. każdy dokument przedstawiamy jako wektor w przestrzeni słów
3. zapytanie przedstawiamy jako wektor w przestrzeni słów
4. liczymy odległości cosinusowe pomiędzy wektorem zapytania a wektorami dokumentów
5. sortujemy dokumenty malejąco według odległości ich wektorów od wektora zapytania

uproszczony wzór na score dokumentu:

$$score(q,d)=\frac{\sum_{s\in q\cap d}((1+\log(tf_{s,q}))\times idf_s)((1+\log(tf_{s,d}))\times idf_s)}{\sqrt{\sum_{s\in q}((1+\log(tf_{s,q}))\times idf_s)^2}\sqrt{\sum_{s\in d}((1+\log(tf_{s,d}))\times idf_s)^2}}$$

Vector Space Model w praktyce

scoring Lucene:

$$score(q,d) = \frac{|q \cap d|}{|q|} \cdot querynorm(q) \cdot \sum_{t \in q} (tf_{t,d} \cdot idf_t^2 \cdot boost(s) \cdot norm(t,d))$$

gdzie:

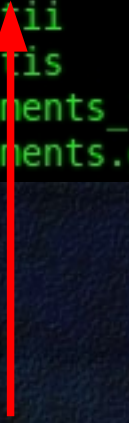
- *querynorm(q)* - czynnik normalizujący (stała), wprowadzony aby dało się porównać wyniki różnych typów zapytań, nie ma wpływu na pozycję dokumentu na liście
- *tf* – zmodyfikowane *term frequency* $tf_{t,d} := \sqrt{tf_{t,d}}$
- *idf* – zmodyfikowane *inverse document frequency* $idf_t := 1 + \log \frac{N}{df + 1}$
- *boost(s)* – dodatkowa waga słowa (termu) ustawiana w treści zapytania
- *norm(t, d) – norma dokumentu,* liczba obliczana w trakcie indeksowania i zapisywana w indeksie:

$$norm(t,d) = boost(d) \cdot \frac{1}{\sqrt{|field|}} \cdot boost(field)$$

  - *boost(d)* – dodatkowa waga dokumentu, ustawiana w czasie indeksowania
  - |*field*| - długość pola (liczba słów)
  - *boost(field)* – dodatkowa waga pola, ustawiana w czasie indeksowania

Vector Space Model w praktyce

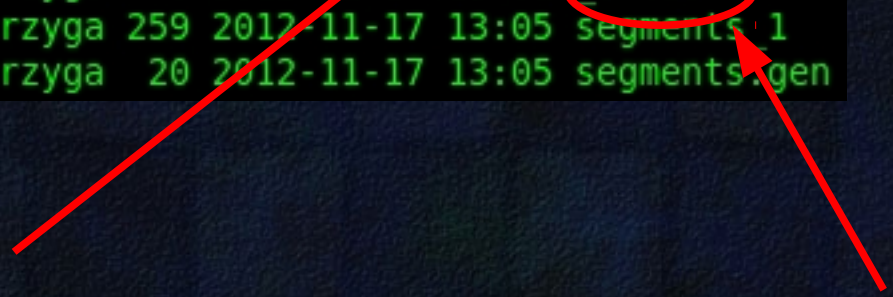
```
-rw-r--r-- 1 strzyga strzyga  49 2012-11-17 13:05 _0.fdt
-rw-r--r-- 1 strzyga strzyga  28 2012-11-17 13:05 _0.fdx
-rw-r--r-- 1 strzyga strzyga  15 2012-11-17 13:05 _0.fnm
-rw-r--r-- 1 strzyga strzyga   9 2012-11-17 13:05 _0.frq
-rw-r--r-- 1 strzyga strzyga   7 2012-11-17 13:05 _0.nrm
-rw-r--r-- 1 strzyga strzyga   9 2012-11-17 13:05 _0.prx
-rw-r--r-- 1 strzyga strzyga  35 2012-11-17 13:05 _0.tii
-rw-r--r-- 1 strzyga strzyga  79 2012-11-17 13:05 _0.tis
-rw-r--r-- 1 strzyga strzyga 259 2012-11-17 13:05 segments_1
-rw-r--r-- 1 strzyga strzyga  20 2012-11-17 13:05 segments.gen
```



normy dokumentów

Vector Space Model w praktyce

```
-rw-r--r-- 1 strzyga strzyga  49 2012-11-17 13:05 _0.fdt
-rw-r--r-- 1 strzyga strzyga  28 2012-11-17 13:05 _0.fdx
-rw-r--r-- 1 strzyga strzyga  15 2012-11-17 13:05 _0.fnm
-rw-r--r-- 1 strzyga strzyga   9 2012-11-17 13:05 _0.frq
-rw-r--r-- 1 strzyga strzyga   7 2012-11-17 13:05 _0.nrm
-rw-r--r-- 1 strzyga strzyga   9 2012-11-17 13:05 _0.prx
-rw-r--r-- 1 strzyga strzyga  35 2012-11-17 13:05 _0.tii
-rw-r--r-- 1 strzyga strzyga  79 2012-11-17 13:05 _0.tis
-rw-r--r-- 1 strzyga strzyga 259 2012-11-17 13:05 segments_1
-rw-r--r-- 1 strzyga strzyga  20 2012-11-17 13:05 segments.gen
```

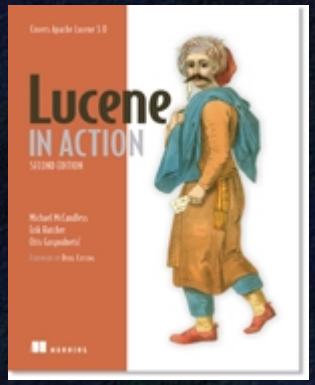


wartości *tf* dla
każdego termu

słownik +
df dla każdego termu

Zasoby wiedzy

- Lucene Wiki: http://wiki.apache.org/lucene-java
- Luke http://code.google.com/p/luke/
- Java User List: java-user@lucene.apache.org
- M. McCandless, E. Hatcher, O. Gospodnetić *”Lucene in Action, Second Edition”*



- Lucid Works:http://www.lucidworks.com/
- C.D. Manning, P. Raghavan, H. Schutze *„Introduction to Information Retrieval” http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html*