Indeksowanie i wyszukiwanie

Eksploracja danych w Internecie

- **Index**: struktura danych tworzona na podstawie tekstu aby przyspieszyć wyszukiwanie

- Wydajność indeksowania w systemach IR mierzy się przez:
  - **czas indeksowania:** czas niezbędny do budowy indeksu,
  - **przestrzeń indeksowania:** przestrzeń pamięci używana podczas generacji indeksu
  - **pamięć indeksu:** pamięć wymagana do zapisania indeksu
  - **opóźnienie zapytania (latency):** czas pomiędzy pojawieniem się zapytania i generacją odpowiedzi
  - **przepustowość zapytania:** średnia ilość pytań przetwarzanych na sekundę

- Podczas modyfikacji tekstu indeks także podlega zmianom

Eksploracja danych w Internecie

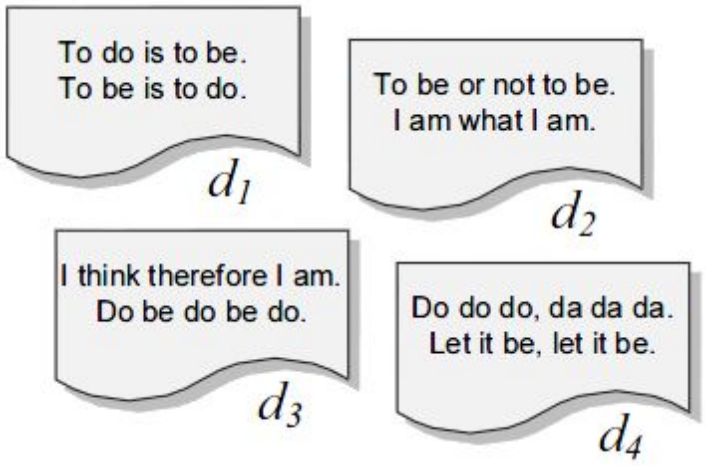
- Bieżąca technologia indeksowania nie jest dobrze przystosowana do bardzo częstych zmian w zbiorze tekstów
- Zbiory pół-statyczne (*semi-static*) są modyfikowane w regularnych, stałych przedziałach czasowych (np. raz dziennie)
- Większość zbiorów tekstowych w sieci jest indeksowana pół-statycznie

- **Indeks odwrócony**: mechanizm indeksowania zbiorów tekstowych przyspieszający wyszukiwanie
- Struktura tego indeksu jest złożona z 2 elementów: **słownika** i **wystąpień** (*occurences*)
- Słownik: zbiór różnych słów tekstu; dla każdego elementu słownika indeks zapamiętuje zawierające go dokumenty (indeks odwrócony)

Eksploracja danych w Internecie

- Macierz **term-dokument** jest rzadka i wymaga dużo pamięci do przechowywania

Vocabulary	n_i	d_1	d_2	d_3	d_4
to	2	4	2	-	-
do	3	2	-	3	3
is	1	2	-	-	-
be	4	2	2	2	2
or	1	-	1	-	-
not	1	-	1	-	-
I	2	-	2	2	-
am	2	-	2	1	-
what	1	-	1	-	-
think	1	-	-	1	-
therefore	1	-	-	1	-
da	1	-	-	-	3
let	1	-	-	-	2
it	1	-	-	-	2

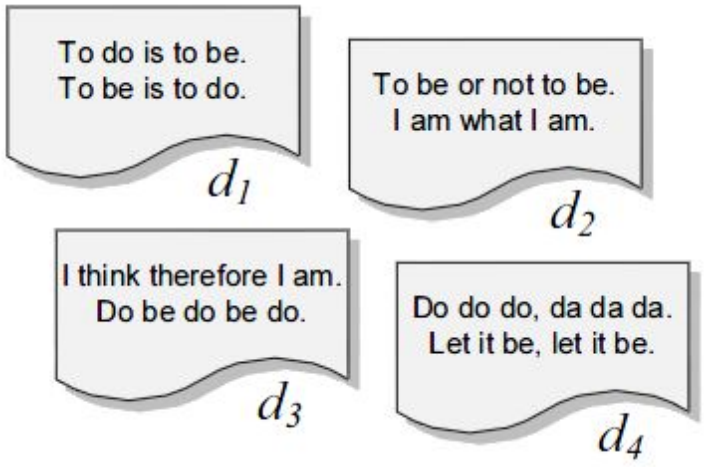


Należy z każdym elementem słownika powiązać listę dokumentów; zbiór takich list to wystąpienia (*occurences*)

Eksploracja danych w Internecie

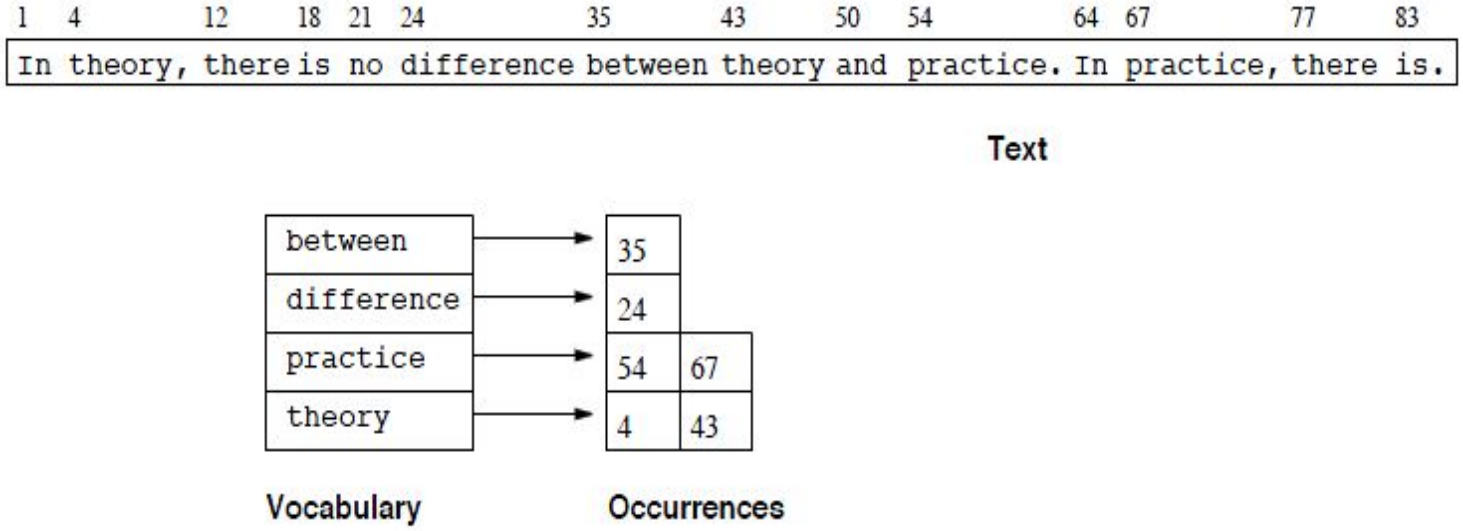
- Podstawowy indeks odwrotny – zbiór list zawierających identyfikatory dokumentów i ilości wystąpień :

Vocabulary	n_i	Occurrences as inverted lists
to	2	[1,4],[2,2]
do	3	[1,2],[3,3],[4,3]
is	1	[1,2]
be	4	[1,2],[2,2],[3,2],[4,2]
or	1	[2,1]
not	1	[2,1]
I	2	[2,2],[3,2]
am	2	[2,2],[3,1]
what	1	[2,1]
think	1	[3,1]
therefore	1	[3,1]
da	1	[4,3]
let	1	[4,2]
it	1	[4,2]



Indeksowanie i wyszukiwanie

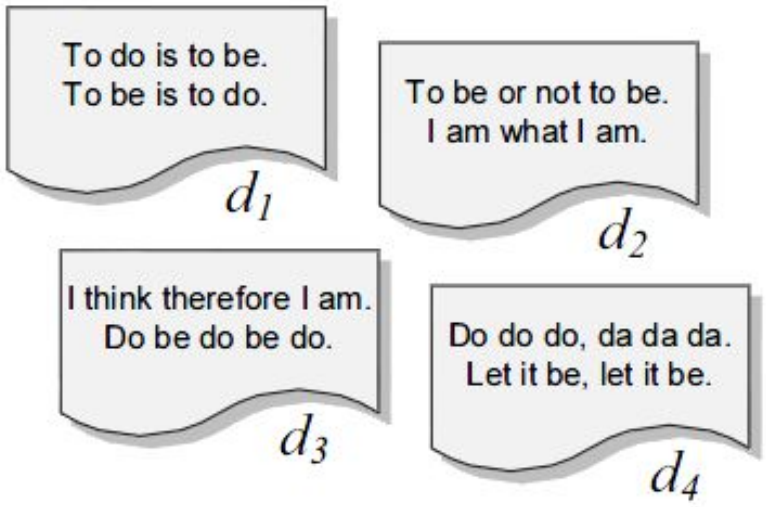
- Podstawowy indeks odwrotny nie jest wystarczający do wyszukiwania odpowiedzi na pytania ze zwrotami (phrase) lub przybliżone (proximity queries)
- W tym celu należy go uzupełnić o pozycję każdego słowa w każdym dokumencie tworząc pełny indeks odwrotny



Indeksowanie i wyszukiwanie

- Pełny indeks odwrotny (adresujący słowa) – przykład:

Vocabulary	n_i	Occurrences as full inverted lists
to	2	[1,4,[1,4,6,9]],[2,2,[1,5]]
do	3	[1,2,[2,10]],[3,3,[6,8,10]],[4,3,[1,2,3]]
is	1	[1,2,[3,8]]
be	4	[1,2,[5,7]],[2,2,[2,6]],[3,2,[7,9]],[4,2,[9,12]]
or	1	[2,1,[3]]
not	1	[2,1,[4]]
I	2	[2,2,[7,10]],[3,2,[1,4]]
am	2	[2,2,[8,11]],[3,1,[5]]
what	1	[2,1,[9]]
think	1	[3,1,[2]]
therefore	1	[3,1,[3]]
da	1	[4,3,[4,5,6]]
let	1	[4,2,[7,10]]
it	1	[4,2,[8,11]]



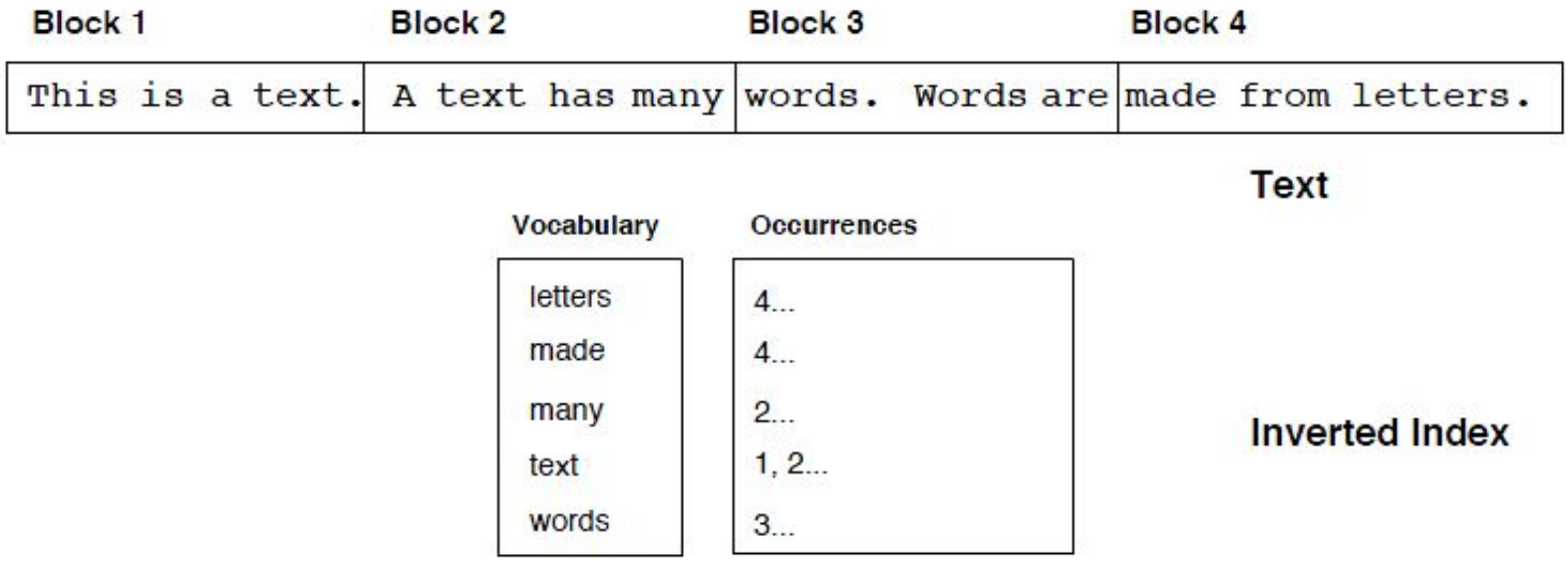
Indeksowanie i wyszukiwanie

- Przestrzeń zajmowana przez słownik jest relatywnie mała w porównaniu z rozmiarem tekstu; rośnie jak $O(n^{\beta})$, gdzie $\beta \epsilon [0.4, 0.6]$, n – rozmiar zbioru.
- We wzorcowej kolekcji dokumentów TREC-3 1GB dokumentów posiada słownik rzędu zajmujący ok. 5MB
- Słownik można dodatkowo redukować poprzez różne techniki np. *stemming*.
- Indeksy adresujące tylko dokumenty typowo zajmują 20% do 40% rozmiaru tekstu, kiedy usuwa się tzw. wyrazy nieistotne (*stopwords*).

Indeksowanie i wyszukiwanie

- Technika adresowania blokowego ograniczająca wymagania przestrzeni adresowej

- Dokument dzieli się na bloki i zapisuje tylko adresy bloków zawierające wyrazy, zamiast adresów poszczególnych wyrazów



Indeksowanie i wyszukiwanie

- Przykład: przestrzeń zajmowana przez indeksy odwrotne zależnie od rozmiaru dokumentów ; w każdej kolekcji lewa kolumna indeksuje tylko wyrazy istotne, prawa wszystke wyrazy

Index granularity	Single document (1 MB)		Small collection (200 MB)		Medium collection (2 GB)	
Addressing words	45%	73%	36%	64%	35%	63%
Addressing documents	19%	26%	18%	32%	26%	47%
Addressing 64K blocks	27%	41%	18%	32%	5%	9%
Addressing 256 blocks	18%	25%	1.7%	2.4%	0.5%	0.7%

Indeksowanie i wyszukiwanie

- Podział tekstu na niewielkie bloki stałej wielkości zwiększa efektywność wyszukiwania – długie bloki są częściej wybierane i ich przeszukiwanie trwa dłużej

- Powtarzające się referencje do tych samych słów, w tym samym kontekście sprowadza się do jednej referencji

Indeksowanie i wyszukiwanie

- Zapytania w formie pojedynczych słów (single word):

- przeszukiwanie słownika może być prowadzone z użyciem haszowania, drzew trie lub B-drzew,
- Pierwsze dwie metody dają koszt wyszukiwania O(m), gdzie m – długość zapytania
- Przyjmuje się, że słownik pozostaje w pamięci operacyjnej a lista wystąpień jest pobierana z dysku

Indeksowanie i wyszukiwanie

- Zapytania w formie wielu słów (multiple word):
- **Pytania koniunktywne (AND)**: poszukują wszystkich słów zapytania z listą indeksu odwrotnego dla każdego z nich. Poszukuje się iloczynu zbiorów list odwrotnych.
- **Pytania dysjunktywne (OR)**: skleja się listy indeksów odwrotnych poszczególnych słów

- Najbardziej czasochłonna operacja na indeksach odwrotnych to łączenie list wystąpień
- m, n –rozmiary dwóch rozpatrywanych list w pamięci szeregowej
  - m<<n to najlepiej m razy przeszukać listę n elementów aby dokonać wstawienia
  - $m \approx n$ można zastosować dwukrotnie algorytm przeszukiwania binarnego – wymaga Średnio m+n porównań

Indeksowanie i wyszukiwanie

- Pytania kontekstowe (frazy i zapytania przybliżone):

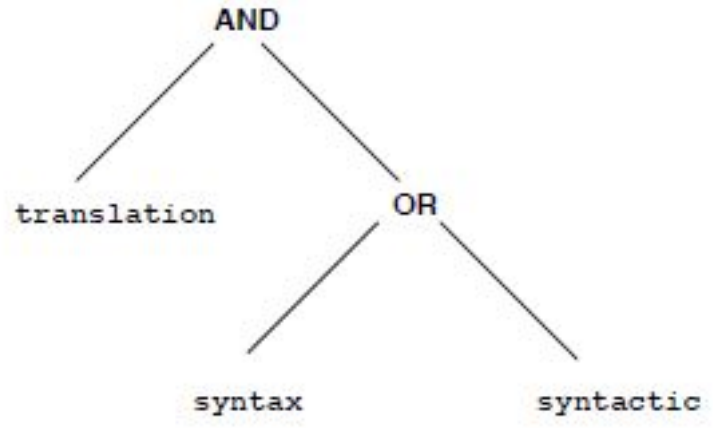
- listy elementów są przeszukiwane aby znaleźć pozycje gdzie pojawia się ciągła sekwencja słów (frazy) lub sekwencja słów dostatecznie blisko siebie (pytania przybliżone)

- Stosuje się algorytmy podobne jak przy dwóch łączeniu list

- Dla wyszukania fraz można stosować także indeksowanie par słów przy pomocy zbliżonych algorytmów

Indeksowanie i wyszukiwanie

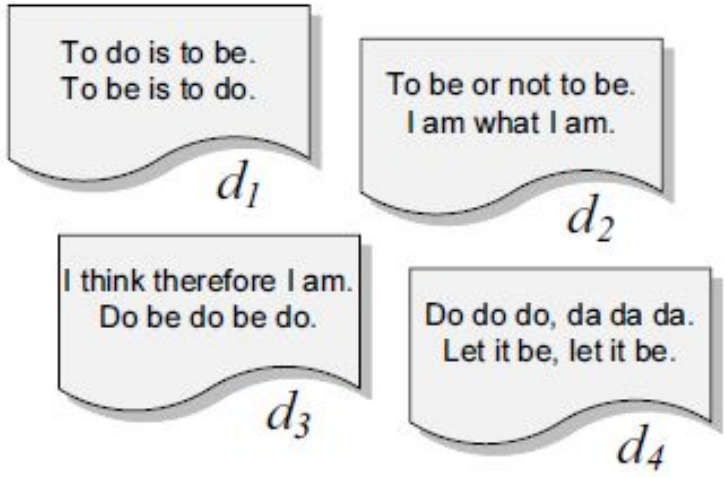
- Zapytania boolowskie: - składnia w formie drzewa



- w pierwszej fazie określa się które dokumenty są do porównania,
- w drugiej fazie ocenia się istotność dokumentów,
- w trzeciej fazie wyznacza się dokładne pozycje dopasowania

Indeksy odwrotne - ranking

- Jak znaleźć k pierwszych dokumentów mając listy odwrotne posortowane według wag?
- W przypadku pojedynczego słowa zapytania sortowanie już zostało wykonane
- Dla innych zapytań trzeba skleić odpowiednie listy
- Np. poszukujemy odpowiedzi na pytanie dysjunktywne „ to do” w kolekcji dokumentów



Indeksy odwrotne - ranking

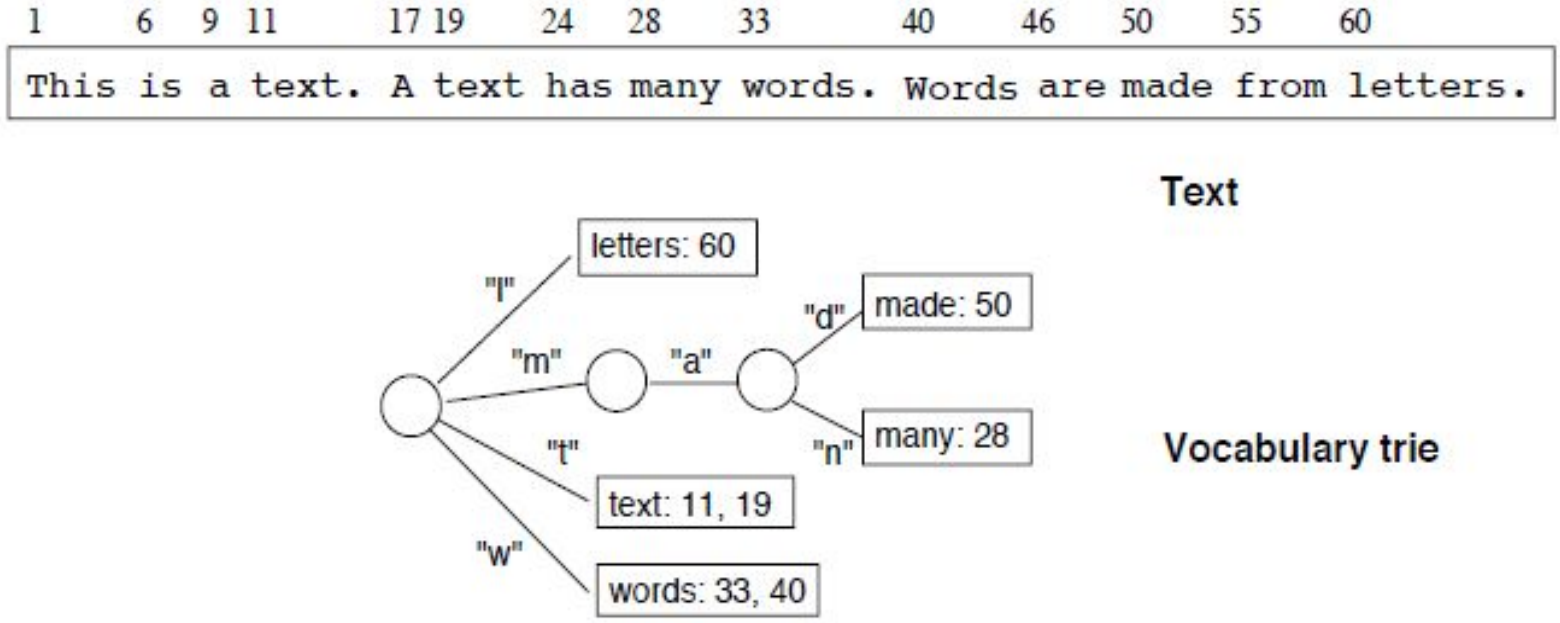
- W najprostszym i najkosztowniejszym podejściu wykonuje się ranking wszystkich dokumentów np. w modelu wektorowym aby wybrać te najbardziej istotne.
- Przy wykorzystaniu indeksu odwrotnego można zmaksymalizować iloczyn TF-IDF używając następującego podejścia heurystycznego :
  - Przeszukujemy termy w porządku malejącym IDF i wybieramy najpierw „to” o większej wartości IDF,
  - przeszukujemy listę odwrotną termu „to”wg. malejących TF i wybieramy skończoną ilość najlepszych dokumentów np. dwa
  - przeszukujemy listę odwrotną termu „do” i zastępujemy dokumenty z listy „to” jeżeli wartości TF nowych dokumentów są lepsze od poprzednio wybranych

Indeksy odwrotne - ranking

- Budowa indeksu w pamięci RAM jest prosta i nie wymaga szczególnych nakładów obliczeniowych
- Tworzy się pustą strukturę danych do przechowywania słownika (B-drzewo, tablicę haszującą itp.)
- Podczas skanowania tekstu szuka się bieżącego słowa w słowniku
- Jeśli słowo jest nowe zostaje dodane do słownika przed dalszym przetwarzaniem
- Duża tablica indeksu jest allokowana tam gdzie zapisuje się identyfikatory kolejnych słów tekstu

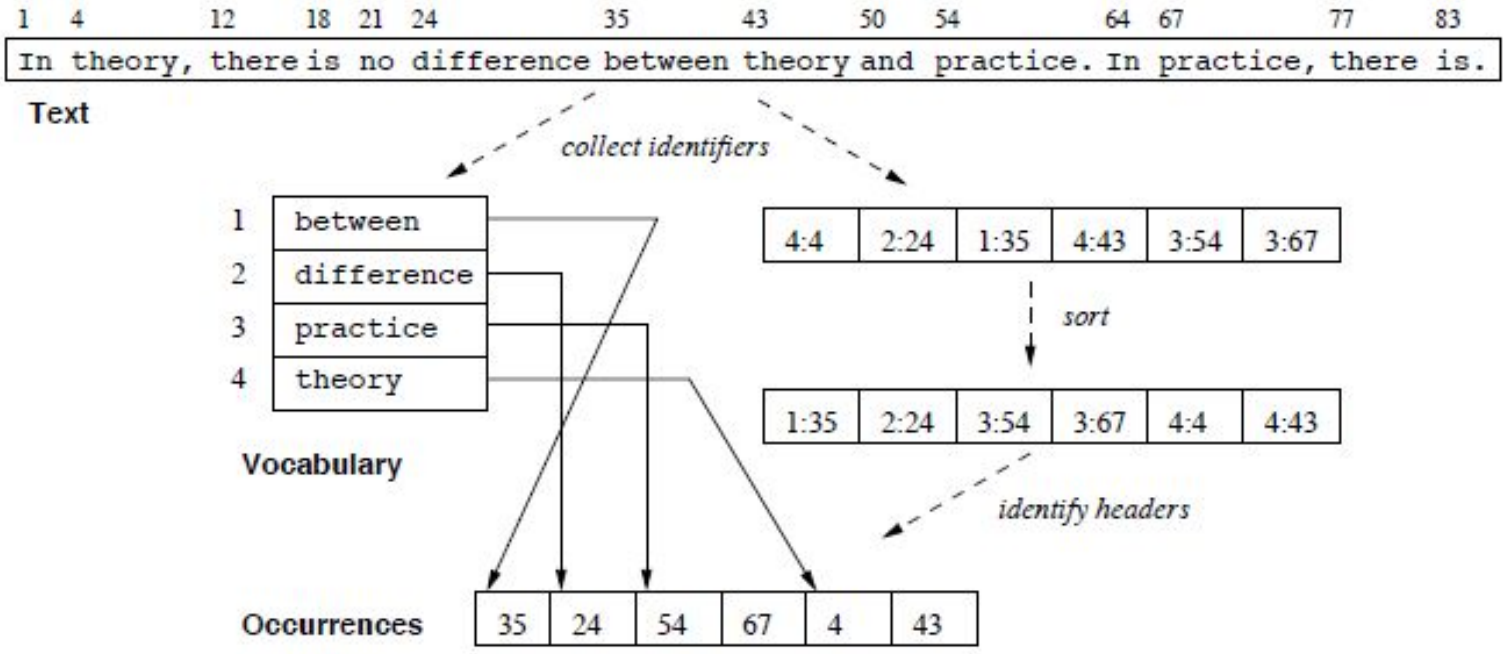
Indeksy odwrotne - ranking

- Pełny indeks odwrotny dla przykładowego tekstu z algorytmem inkrementacyjnym:



Indeksy odwrotne - ranking

- Pełny indeks odwrotny z algorytmem sortującym dla przykładowego tekstu :



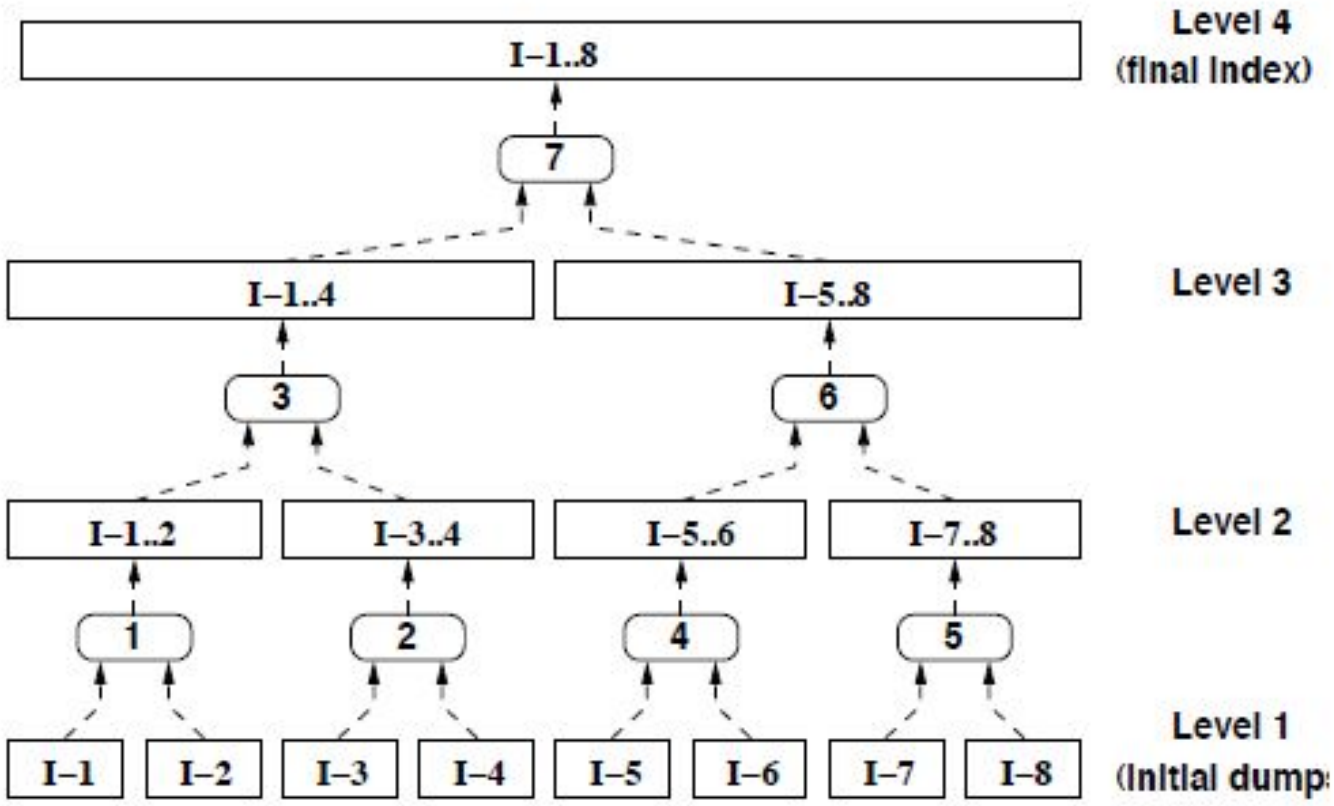
- Aby uniknąć sortowania całości indeksu można tworzyć osobne listy dla każdego wyrazu w słowniku
- Z uwagi na ograniczenia pamięci preferowana jest lista bloków, każdy z wieloma elementami

Indeksy odwrotne - ranking

- Kiedy zakończy się tworzenie listy wystąpień słownik i ta lista są zapisane w różnych plikach na dysku
- Słownik zawiera tylko wskaźniki do list odwrotnych dla każdego ze słów, co pozwala trzymać słownik w pamięci głównej (RAM)
- Po wyczerpaniu pamięci głównej uzyskany indeks częściowy jest przenoszony z pamięci głównej do dysku
- Takie indeksy częściowe są potem składane w sposób hierarchiczny

Indeksy odwrotne - ranking

- Składanie binarne indeksów cząstkowych:



Indeksy odwrotne - ranking

- Indeks odwrócony można aktualizować na trzy sposoby:
  - przez przebudowywanie – gdy badany tekst nie jest zbyt długi,
  - przez przyrostowe aktualizacje – podczas przeszukiwania indeksu gdy wymagane są zmiany,
  - przez fragmentaryczne składanie – nowe dokumenty są oddzielnie indeksowane i ich indeks cząstkowy jest składany z indeksem głównym (najlepszy sposób w ogólnym przypadku)

Drzewa i tablice przyrostkowe

- Do implementacji systemów IR preferowane są indeksy odwrotne
- Sprawdzają się one gdy słowniki nie są zbyt duże, w przeciwnym razie efektywność ich użycia drastycznie maleje
- Indeksy odwrotne sprawdzają się w przypadku języków zachodnich, ale nie w językach niemieckim lub fińskim, które łączą krótkie sylaby w długie słowa; nie sprawdzają się także w językach dalekowschodnich
- Nie pyta się o słowa składane, ale o poszczególne składniki je tworzące
- W takich przypadkach używa się metody drzew i tablic przyrostkowych

Drzewa i tablice przyrostkowe

- Drzewa i tablice przyrostkowe umożliwiają przeszukiwanie indeksowe dowolnych podłańcuchów pasujących do łańcucha zapytania,
- Indeksy traktują tekst jako jeden długi łańcuch a każdy znak w tekście jest traktowany jako przyrostek tekstowy (text **suffix**
- Przykład: suffiksy w tekście „missing missisipi”

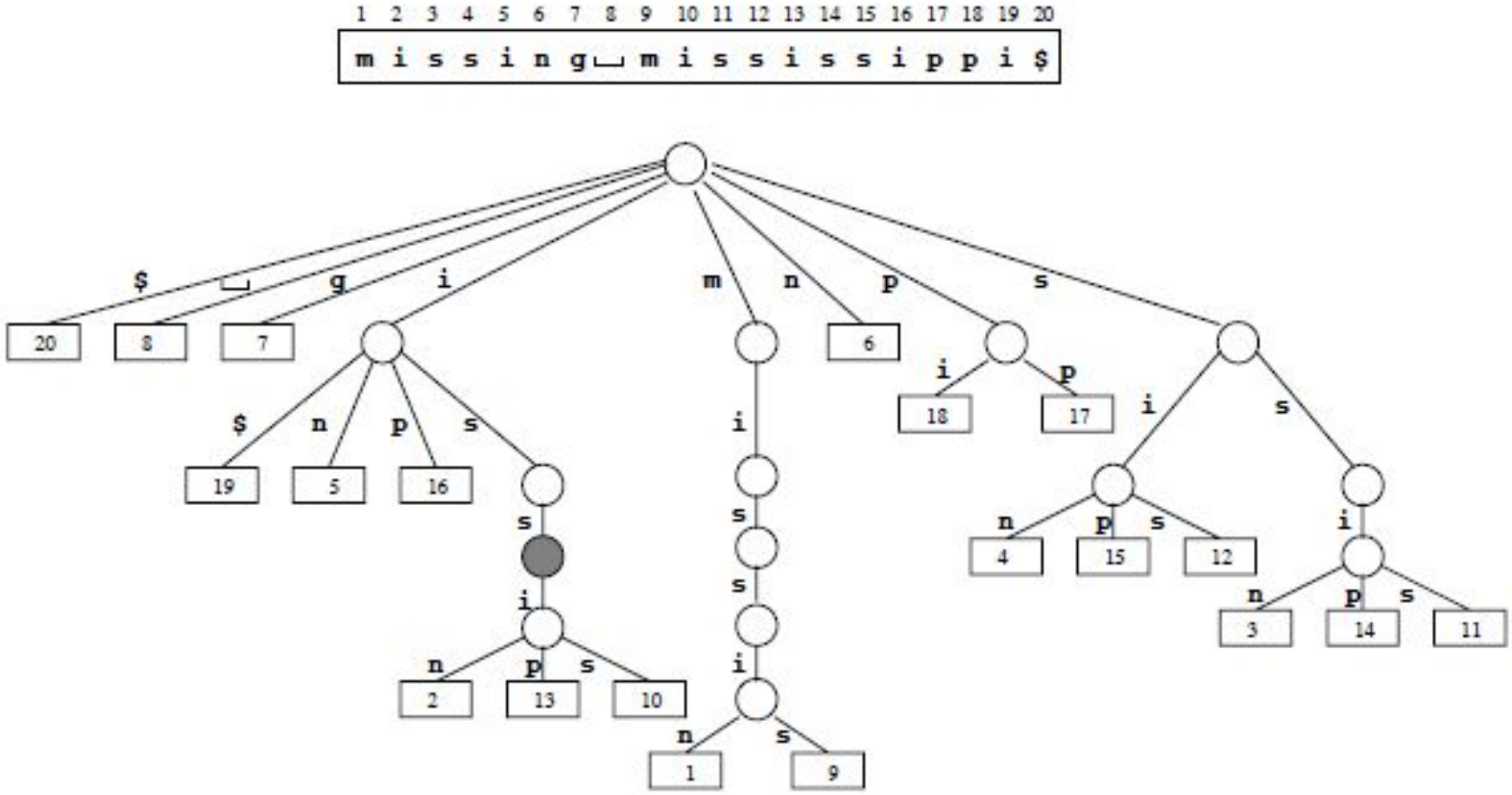
```
missing mississippi
issing mississippi
ssing mississippi
..
ppi
pi
i
```

Drzewa i tablice przyrostkowe

- Strukturę przyrostków można opisać w oparciu o przyrostkowe drzewo wyszukiwania (*suffix trie - trie od re**trie**val*)
- Drzewa przyrostkowe to struktury danych, które zapisują zbiory łańcuchów tak, że można odtworzyć dowolny łańcuch w czasie proporcjonalnym do jego długości, niezależnie od ilości zapisanych łańcuchów
- Zbiór przyrostków $P = \{P_1, \dots P_r\}$ zapisanych jako drzewo wyszukiwania stanowi deterministyczny automat skończony (DFA) rozpoznający $P_1 | \dots | P_r$; poszukiwanie łańcucha przyrostkowego *P* jest równoznaczne z rozpoznaniem tego łańcucha przez DFA
- Drzewo przyrostkowe w tej wersji to struktura danych trie zawierająca wszystkie przyrostki tekstu: $T = t_1 t_1 \dots t_n$,'$'
- Liście drzewa trie zawierają wskaźniki do przyrostków $t_1 t_1 \dots t_n$

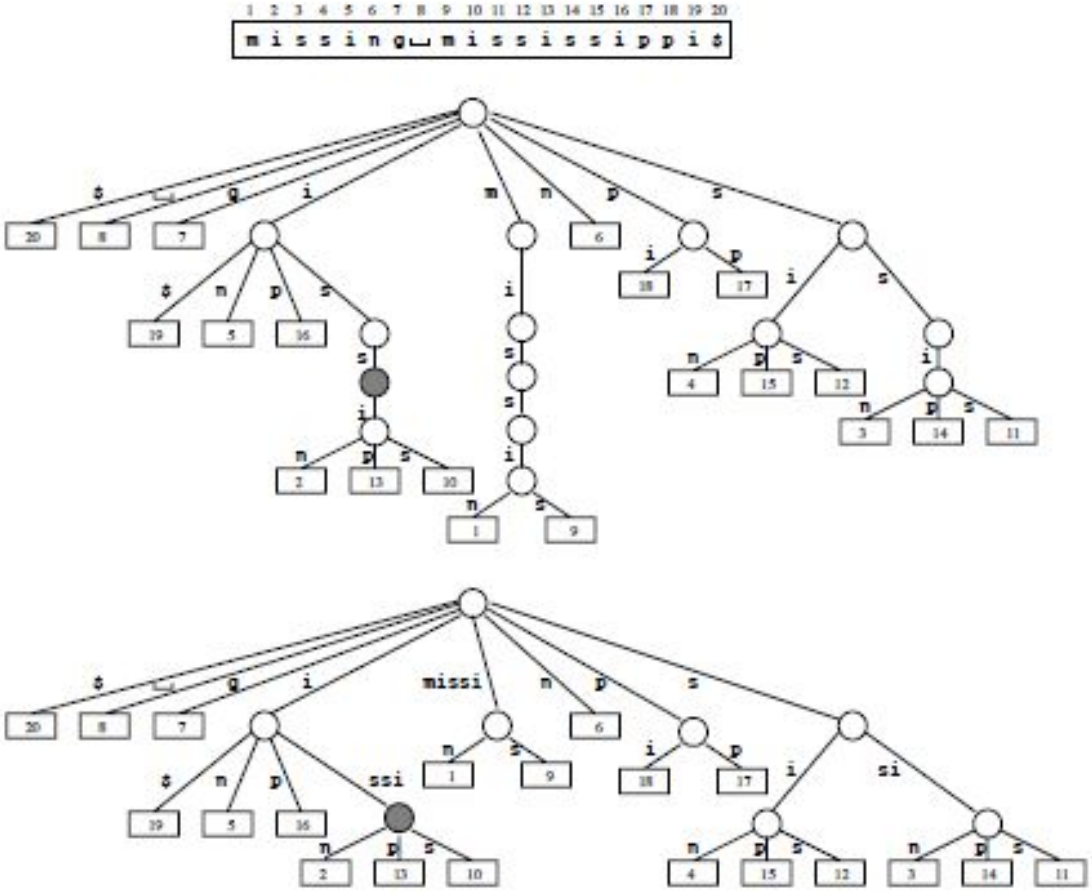
Drzewa i tablice przyrostkowe

- Drzewo wyszukiwania dla tekstu „missing missisipi”



Drzewa i tablice przyrostkowe

- Normalne drzewo przyrostkowe – krawędzie bez rozgałęzień są wyeliminowane



Drzewa i tablice przyrostkowe

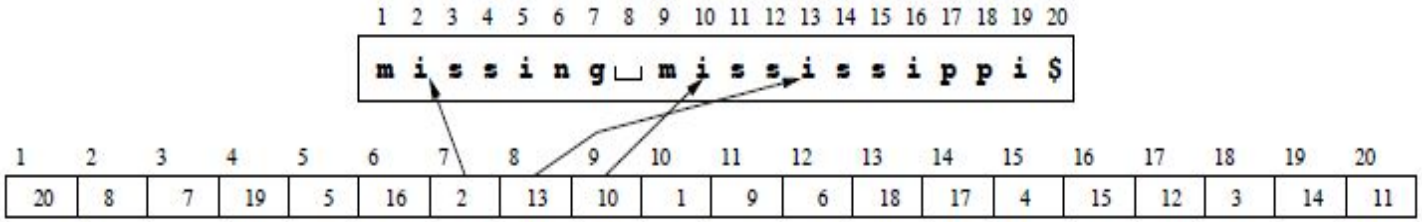
- Drzewa przyrostkowe zajmują dużo miejsca w pamięci
- Zależnie od implementacji zajmują 10 do 20 razy więcej miejsca niż pierwotny tekst
- Ponadto indeksy jako drzewa przyrostkowe są efektywne przede wszystkim w pamięci RAM

- **Tablica przyrostkowa** dla tekstu *T* jest definiowana jako tablica wskaźników na wszystkie przyrostki *T,* które zostały uprzednio wysortowane leksykograficznie (jako liście drzewa przyrostkowego od lewej do prawej)

Drzewa i tablice przyrostkowe

- Tablice przyrostkowe mają podobną funkcjonalność jak drzewa ale mniejsze wymagania pamięciowe – typowo są 4 razy większe od indeksowanego tekstu
- Tablice przyrostkowe nieco mają dłuższe czasy dostępu niż drzewa podczas wyszukiwania

- Tablica przyrostków dla tekstu „missing missisipi”



Drzewa i tablice przyrostkowe

- **Poszukiwanie łańcuchów**:
- Dotyczy wszystkich łańcuchów (części przedrostkowych) pasujących do przyrostka $P = p_1 p_1 \ldots p_m$
- Polega na przesuwaniu się w drzewie wyszukiwania po kolejnych znakach *P*.
- Możliwe są 3 sytuacje:
  - P nie pojawia się w tekście T – nie ma ścieżki odpowiadającej P w drzewie wyszukiwania
  - P zostaje znalezione przed osiągnięciem liścia w gałęzi drzewa – pojawi się na wszystkich pozycjach liści związanych z jego gałęzią
  - Liść drzewa pojawi się przed zakończeniem łańcucha P; należy kontynuować porównywanie tekstu aż do liścia aby zweryfikować obecność P

Drzewa i tablice przyrostkowe

- Jeżeli poszukiwanie dotyczy normalnego drzewa przyrostkowego krawędzie są skojarzone z całymi podłańcuchami
- Wszystkie etykiety krawędzi odchodzące od danego węzła różnią się pierwszym znakiem

Drzewa i tablice przyrostkowe

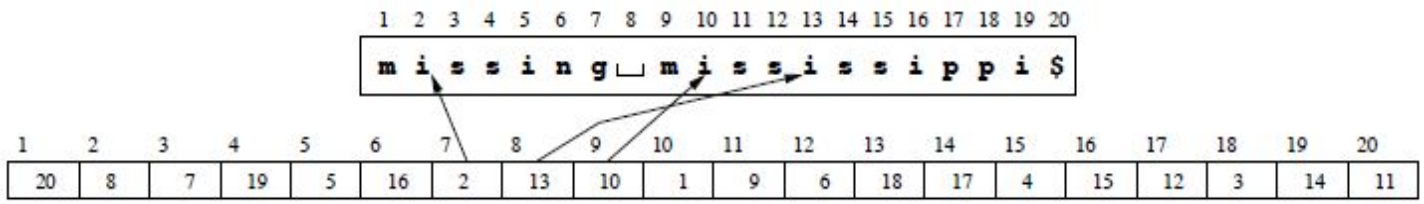
- Pseudokod przeszukiwania drzewa przyrostkowego:

Suffix-Tree-Search $(S,\ P = p_1 p_2 \ldots p_m)$

(1) $i \leftarrow 1$
(2) **while** $true$ **do**
(3) **if** S is a leaf pointing to j **then**
(4) **if** $p_i \ldots p_m = t_{j+i-1} \ldots t_{j+m-1}$
(5) **then return** S
(6) **else return** $null$
(7) **if** there is an edge $S \xrightarrow{p'_1 \cdots p'_s} S' \ \wedge \ p'_1 = p_i$ **then**
(8) $j \leftarrow 0$
(9) **while** $j < s \ \wedge \ i + j \leq m \ \wedge \ p'_{j+1} = p_{i+j}$ **do** $j \leftarrow j + 1$
(10) $i \leftarrow i + j$
(11) **if** $i > m$ **then return** S'
(12) **if** $j < s$ **then return** $null$
(13) $S \leftarrow S'$
(14) **else return** $null$

Drzewa i tablice przyrostkowe

- Przeszukiwanie tablicy przyrostkowej jest typu binarnego z pośrednimi porównaniami
- Każdy krok binarnego wyszukiwania wymaga porównania *P* z przyrostkiem tekstu

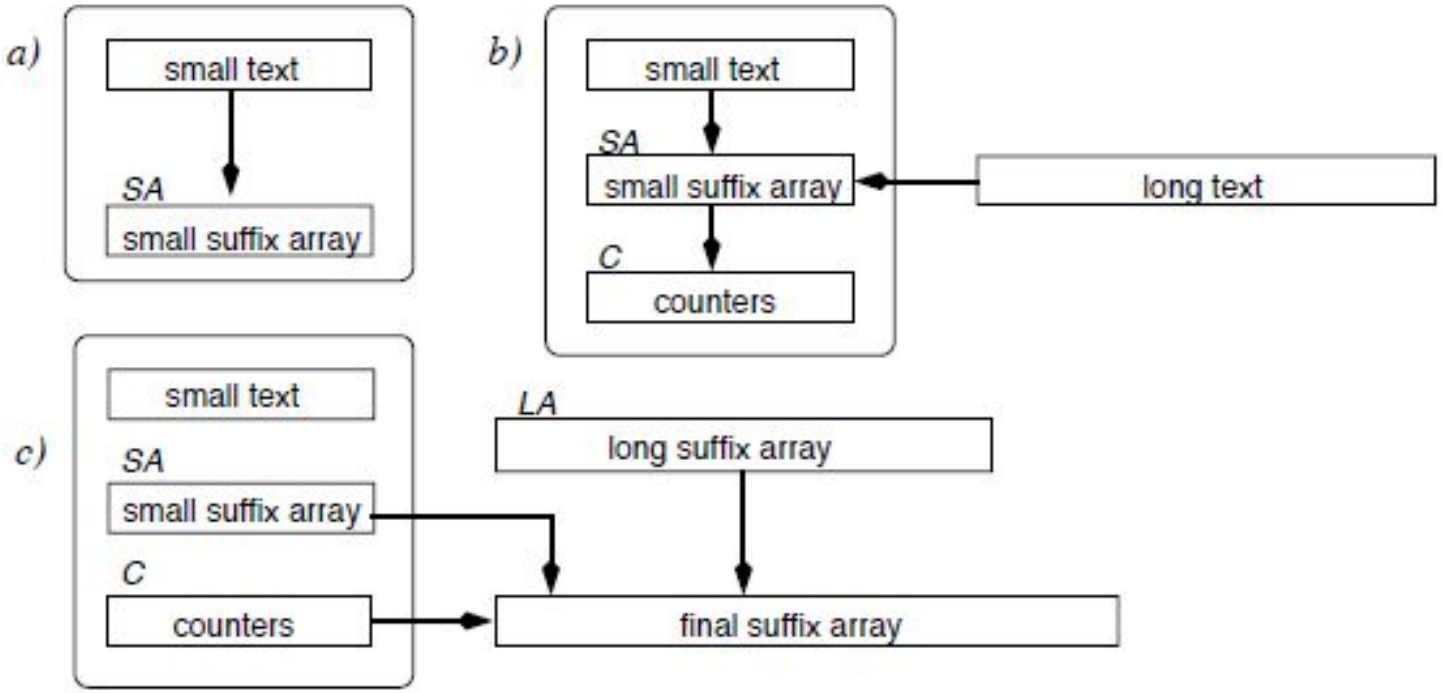


Drzewa i tablice przyrostkowe

- **Drzewa i tablice przyrostkowe dla dużych tekstów**
- Kiedy dane tablic przyrostkowych lub tekstu nie mieszczą się w pamięci głównej specjalne algorytmy są wymagane do pracy z pamięcią zewnętrzną
- Tekst dzieli się na bloki, które mogą być sortowane w pamięci głównej
- Dla każdego bloku buduje się tablicę przyrostkową w pamięci głównej i składa się ją z resztą tej tablicy zbudowaną dla poprzednich bloków

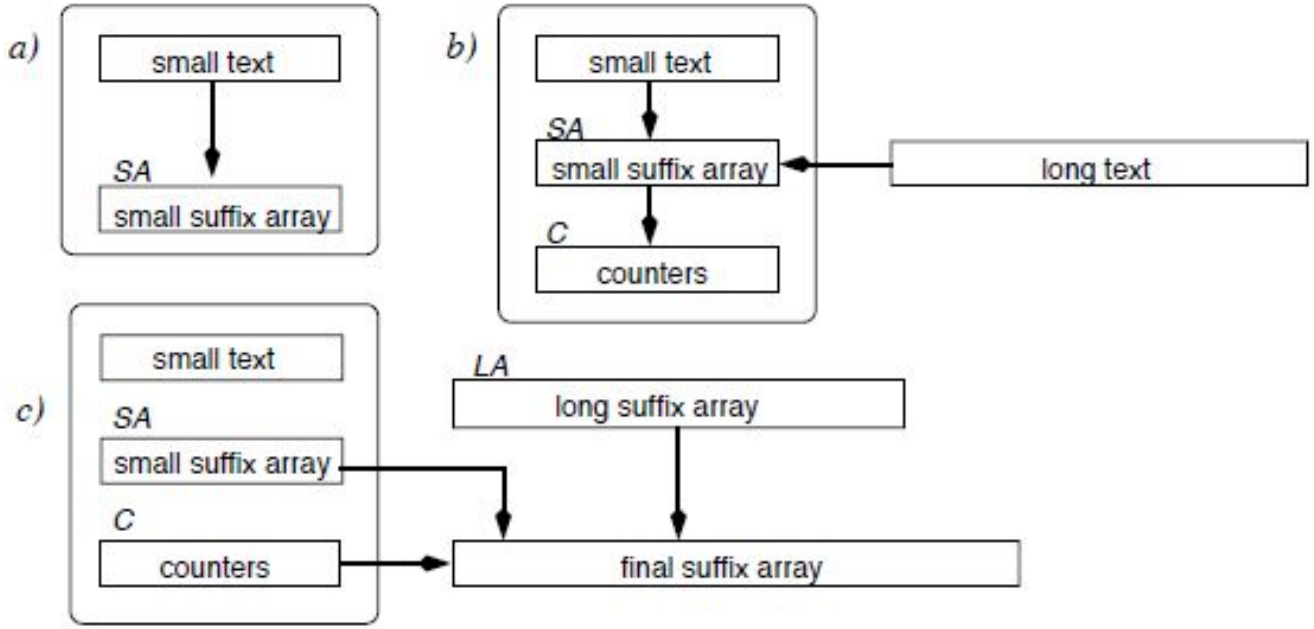
Drzewa i tablice przyrostkowe

- Krok budowy tablicy tablicy przyrostkowej dla dużych tekstów:
  - a) utworzenie małej tablicy przyrostków *SA*,
  - b) wypełnienie tablicy liczników *C*,
  - c) sklejenie tablic *LA* dla bloków 1,2,…,*i*-1 oraz tablicy *SA dla bloku i*



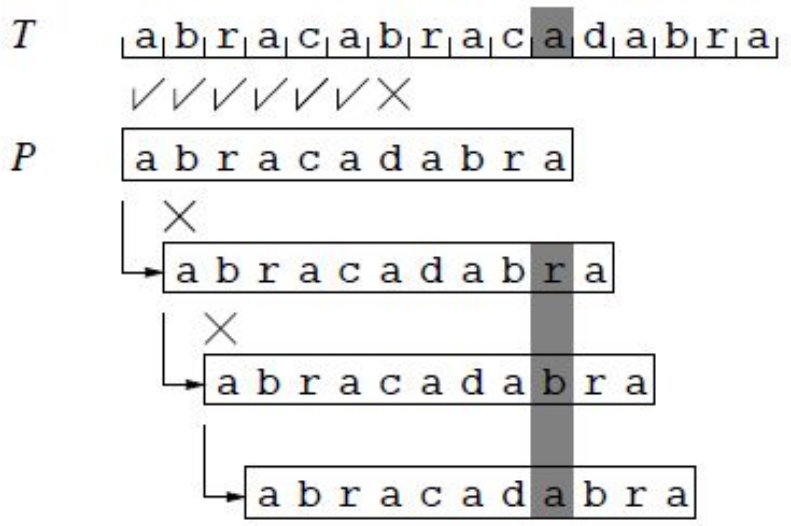
Drzewa i tablice przyrostkowe

- Tablica liczników *C* w każdej komórce *C*[*j*] zapamiętuje ile przyrostków *LA* jest do wstawienia pomiędzy *SA*[*j*] i *SA*[*j+1*]
- Obliczenie *C* nie wymaga dostępu do *LA*; tekst odpowiadający *LA* jest sekwencyjnie wczytywany do pamięci głównej
- Każdy przyrostek tekstu jest poszukiwany w *SA*; jeśli jest pomiędzy *SA*[j] i *SA*[j+1] to inkrementujemy *C*[j]



Wyszukiwanie sekencyjne

- Algorytm „brute force” polega na sprawdzeniu po kolei wszystkich możliwych pozycji wzorca w tekście
- Polega na przesuwaniu wzdłuż łańcucha tekstu okienka o długości m wzdłuż tekstu o długości m; $t_{i+2}...t_{i+m}$ dla $i \leq 0 \leq -m$.
- Wzorzec może wystąpić w każdym z okien, które muszą być po kolei sprawdzone
- *Przykład:*

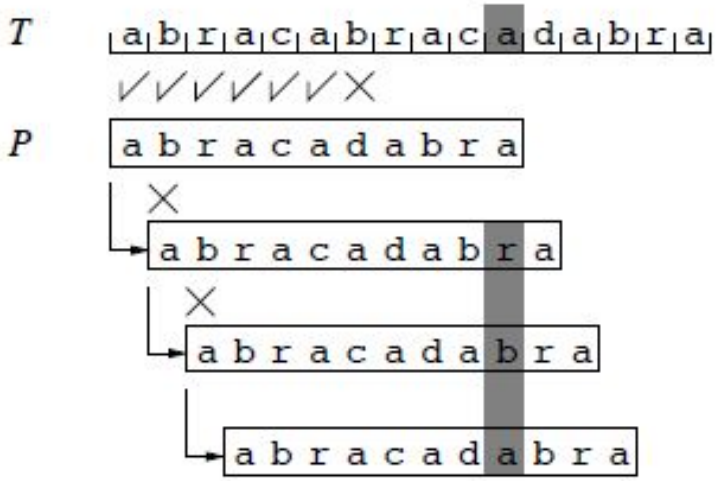


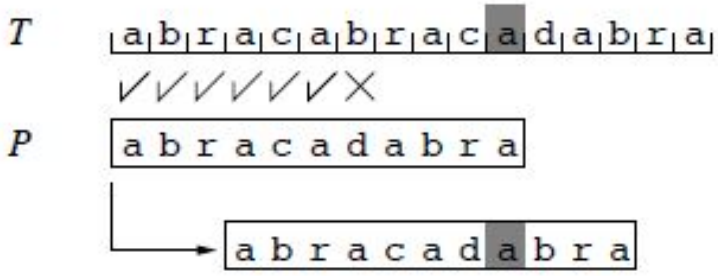
Wyszukiwanie sekencyjne

- Algorytm Horspool'a: najszybszy algorytm wyszukiwania łąńcuchów dla języków naturalnych
- Wykorzystuje pomysł przesuwania okna ze wzorcem jak poprzednio
- Wstępnie przelicza tabelę *d* przesunięć okna indeksowaną znakami alfabetu; *d*[*c*] zapisuje o ile pozycji trzeba przesunąć okno, jeżeli jego ostatnim znakiem jest *c*.
- Inaczej, *d*[*c*] jest odległością od końca wzorca p_m, z wyłączeniem tego końca, do ostatniego wystąpienia *c* w *P*,.

Wyszukiwanie sekencyjne

- Algorytm Horspool'a – przykład





Wyszukiwanie sekencyjne

- Algorytm Horspool'a – pseudokod

Horspool $(T = t_1 t_2 \dots t_n,\ P = p_1 p_2 \dots p_m)$

(1) **for** $c \in \Sigma$ **do** $d[c] \leftarrow m$

(2) **for** $j \leftarrow 1 \dots m-1$ **do** $d[p_j] \leftarrow m - j$

(3) $i \leftarrow 0$

(4) **while** $i \leq n - m$ **do**

(5) $\quad j \leftarrow 1$

(6) $\quad$ **while** $j \leq m \ \wedge\ t_{i+j} = p_j$ **do** $j \leftarrow j + 1$

(7) $\quad$ **if** $j > m$ **then** report an occurrence at text position $i + 1$

(8) $\quad i \leftarrow i + d[t_{i+m}]$

- Przy długich wzorcach i krótkim alfabecie alorytm Horspool'a nie jest efektywny

Wyszukiwanie sekencyjne

- W przypadku ogólnym można przesuwać okno używając q końcowych znaków, a nie tylko ostatniego; jaka byłaby najlepsza wartość q?

- Jeżeli σ oznacza rozmiar alfabetu to wartość przesunięcia $\pi\sigma^q < m$, stąd $q = log_\sigma(m)$ i średni czas szukania wynosi $O(nlog_\sigma(m)/m)$

Wyszukiwanie sekencyjne

- Programowanie dynamiczne:

- Klasyczna metoda przybliżonego dopasowania łańcuchów wskazująca pozycje, gdzie wzorzec *P* pojawia się z co najwyżej *k* błędami.

- Stosuje się różne definicje błędów dopasowania:
  - odległość Hamminga – ilość niezbędnych podstawień znaków w tekście niezbędna dla dopasowania fragmentu tekstu do wzorca
  - odległość edycyjna (Levensteina) – ilość usunięć, wstawień i zamiany znaków w tekście niezbędna dla dopasowania fragmentu tekstu do wzorca

Wyszukiwanie sekencyjne

- Wylicza się tablicę $C[0 \ldots m, 0 \ldots n]$ elementów $C[i, j]$ reprezentujących minimalne ilości *k* błędów przy dopasowaniu wzorca $p_1 p_2 \ldots p_i$ do tekstu $t_1 t_2 \ldots t_j$.
- Tablicę wyznacza się następująco:

$$
\begin{aligned}
C[0, j] &= 0, \\
C[i, 0] &= i, \\
C[i, j] &= \text{if } (p_i = t_j) \text{ then } C[i-1, j-1] \\
&\quad \text{else } 1 + \min(C[i-1, j], C[i, j-1], C[i-1, j-1]),
\end{aligned}
$$

- Dopasowanie pojawia się na pozycjach *j*, takich że $C[m, j] \leq k$.

Wyszukiwanie sekencyjne

Przykład – programowanie dynamiczne poszukujące słowa ‚colour' w tekście ‚kolorama' z k=2 błędami; * opisuje wybraną pozycję wzorca

		k	o	l	o	r	a	m	a
	0	0	0	0	0	0	0	0	0
c	1	1	1	1	1	1	1	1	1
o	2	2	1	2	1	2	2	2	2
l	3	3	2	1	2	2	3	3	3
o	4	4	3	2	1	2	3	4	4
u	5	5	4	3	2	2	3	4	5
r	6	6	5	4	3	2*	3	4	5

Odległość 2* znaków jest spełniona w sensie Hamminga i Levensteina

Drzewa i tablice przyrostkowe

Programowanie dynamiczne w przypadku ogólnym wymaga $O(mn)$ czasu.

W przedstawionym wariancie korzystającym tylko z poprzedzającej kolumny $C[\cdot\,,j\text{-}1]$ do wyliczenia kolumny bieżącej $C[\cdot\,,j]$ algorytm może być implementowany w przestrzeni $O(m)$.

Drzewa i tablice przyrostkowe

Drzewa i tablice przyrostkowe

Drzewa i tablice przyrostkowe