Drzewa sufiksowe

referat z seminarium *zaawansowane struktury danych*
mgr Paweł Rzechonek, II UWR 2012/2013

Borys Dyczko

29 stycznia 2013

Spis treści

1 Wstęp

Wyszukiwanie wzora w tekście jest popularnym problemem algorytmicznym. Istnieją algorytmy, które w szybkim czasie potrafią odpowiedzieć na pytanie o występienia wzorca, np. Boyera-Moore'a, Morrisa-Pratta, Knutha-Morrisa-Pratta. Jednak gdy interesują nas pytania o wiele wzorców zawartych w jednym tekście, powyższe algorytmy się nie sprawdzają. Aby otrzymać lepszą złożność czasową, możemy skorzystać z takich struktur danych jak tablice lub drzewa sufiksowe. Wykorzystują one preprocessing głównego tekstu do budowy swojej struktury, aby w czasie liniowym odpowiedzieć na pytanie o wzorzec.

1.1 Historia

Koncepcję drzew sufiksowych (nazwanych wtedy drzewami pozycyjnymi) wprowadził Weiner w 1973 roku. Konstrukcja została uproszczona przez McCreighta w 1976. Ukkonen w 1995 podał pierwszy liniowy algorytm konstrukcji drzewa sufiksowego online, znany jako Algorytm Ukkonena.

1.2 Definicje

Σ - alfabet
$|w|$ - długość słowa w
$T = t_0t_1t_2...t_{n-1}$ - tekst długości n
$T_i = t_it_{i+1}t_{i+2}...t_{n-1}$ - i-ty sufiks T
n - długość tekstu
m - długość wzorca

Aby ustalić ścisły porządek na sufiksach danego tekstu, będzie wygodnie dołożyć na koniec tekstu znak #. Przyjmujemy, że jest leksykograficznie większy od dowolnego symbolu z Σ.

1.3 Opis

Drzewo sufiksowe jest drzewem, w którym każda ścieżka jest etykietowana kolejnymi symbolami pewnego sufiksu, oraz każdy sufiks T jest obecny w drzewie. Każdej krawędzi jest przypisane jako etykieta pewne podsłowo T. Krawędzie wychodzące z tego samego węzła, różnią się pierwszymi symbolami swoich etykiet.

Etykiety są kodowane przedziałami w tekście T: para liczb $[i, j]$ reprezentuje podsłowo $t_it_{i+1}...t_j$. Dzięki temu reprezentacja ma rozmiar $O(n)$. Wagą krawędzi jest długość odpowiadającego jej słowa.

Przykładowe drzewo sufiksowe znajduje się na rysunku 1.

1.4 Reprezentacja węzła

Różne struktury wykorzystane do reprezentacji węzłów, dają różne złożoności czasowe i pamięciowe. Wybrane struktury są przedstawione w tabeli 1. Idealnym rozwiązaniem wydaje się być tablica haszująca, jednak przy jej wykorzystaniu reprezentacja węzła nie zachowuje porządku. Jest to potrzebne w

Rysunek 1: Drzewo sufiksowe dla słowa BANANA

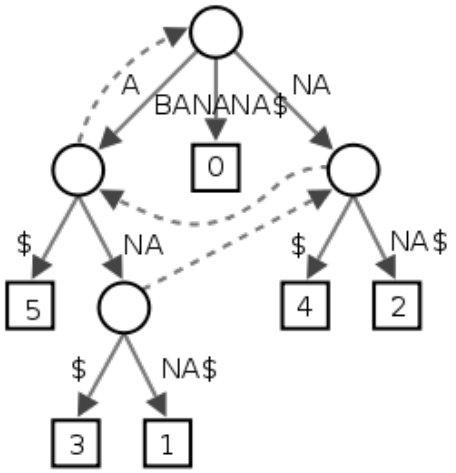


Tabela 1: Reprezentacja węzłów, a złożoność

struktura	czas	pamięć
tablica	$O(m)$	$O(n * \|\Sigma\|)$
BST	$O(m * log\|\Sigma\|)$	$O(n)$
tablica haszująca	$O(m)$	$O(n)$
drzewo Van Emde Boas	$O(m * loglog\|\Sigma\|)$	$O(n)$

niektórych operacjach, np. podczas szukania poprzednika, następnika. Prostym rozwiązaniem w implementacji z małym narzutem na pamięć, jest drzewo BST.

2 Operacje

Wszystkie operacje wykonywane na drzewach sufiksowych są stosunkowo proste, opierają się na umiejętnym przejściu drzewa, z paroma drobnymi modyfikacjami.

Dany jest wzorzec $P = p_0p_1p_2...p_{m-1}$.

2.1 Wyszukiwanie wzorca, ilość wystąpień

Wyszukiwanie wzorca P zaczynamy od korzenia. Za każdym razem przechodzimy po krawędziach w dół drzewa, aż napotkamy miejsce w którym nie ma krawędzi odowiadających literom wzorca. Jeśli podczas przechodzenia drzewa przeczytamy cały wzorzec, oznacza to, że występuje on w tekście. Ilość wystąpien wzorca P odpowiada ilości liści w poddrzewie w którym zakończyliśmy przechodzenie drzewa.

Łatwo udowodnić złożoność czasową $O(m)$ powyższej operacji. Przejść po krawędziach będzie maksymalnie tyle, ile liter ma wzorzec, czyli m. Wykorzystując tablice haszujące, odszukanie krawędzi w węźle zajmuje $O(1)$. Całość sumuje się oczywiście do $O(m)$.

2.2 Pierwsze k wystąpień

Wprowadzenie modyfikacji:

- dodanie do każdego węzła wskaźnika, na lewego, skrajnego syna
- połączenie wszystkich liści w listę cykliczną

Modyfikacje nie podnoszą złożności, a jedyne stałą podczas budowy drzewa sufiksowego, o czym jest w dalszej częsci pracy.

Aby znaleźć pierwsze k wystąpień, należy odszukać odpowiedni węzeł tak samo jak w rozdziale 2.1. Następnie przeglądnąć listę liści. Daje to oczywistą złożoność $O(k + m)$.

2.3 Najdłuższe powtarzające się podsłowo

Operacja polega na znalezienieniu najgłębszego węzła który nie jest liściem. Czas operacji O(n).

Dlaczego jest to poprawne? Węzeł o etykiecie x, który posiada 2 liście, świadczy o istnientu 2 sufiksów w tekście, których prefiksem jest x. Korzystając z tego, że szukamy najgłębszego węzła, dostajemy, że 2 takie prefiksy x, są najdłuższe.

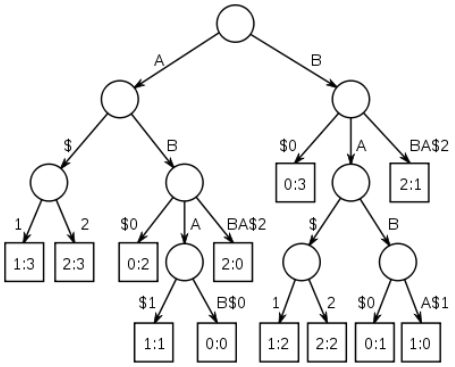
2.4 Wiele tekstów

Danych jest z tekstów T^0, T^2, T^3 ... T^{z-1}. Wprowadzamy modyfikacje w reprezentacji drzewa:

- do każdego tekstu T^i i dołączamy na koniec $\$_i$
- każdy węzeł posiada informację o liczbie unikalnych wystąpień \$ w jego poddrzewie

Przykładowe drzewo sufiksowe dla kilku teksów znajduje się na rysunku 2.

Rysunek 2: Drzewo sufiksowe dla 3 słów: ABAB, BABA, ABBA.



2.4.1 Najdłuższy wspólny podciąg

Znalezienie węzła o największej głębokości, który posiada wszystkie $\$_i$. Dowód podobny jak w rodziale 2.3. Złożoność $O(\sum_{i=0}^{z-1} |T^i|)$.

2.4.2 Liczba tekstów zawierających wzorzec

Przejście drzewa zgodnie z literami wzorca i odczytanie ilości unikalnych \$. Sytuacja analogiczna jak w rozdziale 2.1.

3 Konstrukcja drzew sufiksowych

3.1 Definicje

$Label(v)$ = słowo otrzymane podążając po drzewie od korzenia do węzła v.

$Suf(v)$ (dowiązanie sufiksowe) = v' takie że
$Label(v) = y_0y_1y_2...y_k$
$Label(v') = y_1y_2...y_k$

$Suf(root) = root$.

$parent(v)$ = rodzic v w drzewie T
$parent(root) = root$

3.2 Algorytm McCreighta

Algorytm buduje skompresowane drzewo sufiksowe w kolejności od najdłuższego sufiksu do najkrótszego. Ogólny schemat algorytmu jest następujący:

- W fazie pierwszej inicjujemy drzewo najdłuższym sufiksem, czyli pojedynczą krawędzią reprezentującą cały tekst T.
- Załóżmy, że jesteśmy w fazie $k > 1$ i dodajemy do drzewa $sufiks_k$.
- Przedstawmy słowo $sufiks_k$ jako konkatenację dwóch słów pq takich, że słowo p jest najdłuższym prefiksem $sufiks_k$, który można utworzyć podążając ścieżką od korzenia w dół drzewa. Taki rozkład jest jednoznaczny.
- Definiujemy głowę $head_k$ k-tego sufiksu jako taki węzeł v (być może niejawny) w drzewie, że $Label(v) = p$.
- Dodajemy do drzewa krawędź reprezentującą słowo q w węźle $head_k$.

Szukanie głowy w sposób naiwny - $O(n^2)$
Aby obiniżyć złożoność, skorzystamy z dowiązań sufiksowych. Aby szybko znajdować głowę k-tego sufiksu, korzystamy z następujących własności drzew sufiksowych:

- $head_i$ jest potomkiem $Suf(head_{i-1})$
- $Suf(v)$ jest potomkiem $Suf(parent(v))$, $\forall v \in T$

Skorzystamy z 2 sposobów schodzenia w dół drzewa:
slowFind - przemieszczanie się znak po znaku. Używane, gdy nie wiemy wystarczająco wiele o szukanym słowie, aby przemieszczać się w sposób szybki.
$fastFind$ - gdy wyszukujemy w drzewie słowo, o którym wiemy, że na pewno je znajdziemy, a bardziej interesuje nas gdzie (w jakim węźle jawnym lub

niejawnym) je znajdziemy. Pozwala to w czasie stałym omijać całe krawędzie drzewa.

Funkcje dla argumentów (v, y) zwracają węzeł (być może niejawny), na którym kończy się zejście w dół drzewa zapoczątkowane w węźle v, w poszukiwaniu słowa y.

3.2.1 Algorytm

Pseudokod algorytmu został zaprezentowany jako algorytm 1.

Algorithm 1 Algorytm McCreighta - konstrukcja offline

```
T ← inicjalne drzewo, zbudowane z jednej krawędzi etykietowanej całym tekstem x o długości n
head_1 ← root
leaf_1 ← jedyny liść drzewa
for i = 2 → n do
    p ← etykieta na krawędzi parent(head_{i-1}) → head_{i-1}
    q ← etykieta na krawędzi head_{i-1} → leaf_{i-1}
    if head_{i-1} = root then
        Usuń pierwszy znak z q
        head_i ← slowfind(root, q)
    else
        u ← parent(head_{i-1})
        if u = root then
            Usuń pierwszy znak z p
            v ← fastfind(root, p)
        else
            v ← fastfind(Suf(u), p)
        end if
        if v jest węzłem niejawnym then
            head_i ← v
        else
            head_i ← slowfind(v, q)
        end if
    end if
    Suf(head_{i-1}) ← v
    Stwórz węzeł leaf_i
    if head_i jest węzłem niejawnym then
        rozbij krawędź, na której znajduje się head_i na dwie krawędzie
    end if
    Stwórz krawędź head_i → leaf_i o odpowiedniej etykiecie
end for
```

3.2.2 Złożoność

Twierdzenie 1. *Algorytm McCreighta buduje drzewo sufiksowe tekstu o długości n w czasie $O(n)$, przy założeniu $|\Sigma| = O(1)$.*

Dowód. Rozważmy oddzielnie czas działania wszystkich wykonań *slowFind* i $fastFind$. Wszystkie pozostałe operacje działają w czasie stałym. Niech $depth(v)$

oznacza głębokość węzła v w drzewie sufiksowym, tzn. liczbę węzłów w skompresowanym drzewie na ścieżce od korzenia do v.

Operacja $fastFind$ zwiększa głębokość, na której przebywamy. Przejście do rodzica oraz przejście dowiązaniem sufiksowym zmniejsza tę głębokość. Zachodzi $depth(v) <= depth(Suf(v)) + 1$. Oznacza to, że w każdej iteracji algorytmu najpierw zmniejszamy głębokość o co najwyżej 2, a następnie zwiększamy tę głębokość. Nie możemy jej zwiększać w nieskończoność, bo liczba węzłów drzewa jest w danej fazie liniowa względem numeru iteracji, zatem po wszystkich iteracjach wykonamy co najwyżej liniowo wiele przemieszczeń po węzłach drzewa w trakcie $fastfind$.

Czas spędzony na wykonywaniu $slowFind$ także jest liniowy, co zakończy dowód. Przez zapis $|head_i|$ rozumiemy odległość węzła $head_i$ od korzenia w sensie nieskompresowanego drzewa sufiksowego.

Używamy $slowFind$ aby znaleźć $head_i$ rozpoczynając od $Suf(head_{i-1})$. Ponieważ początkowych $|head_{i-1}|$ znaków znajdujemy przy pomocy $fastFind$, praca wykonana przez $slowFind$ wynosi $|head_i| - |head_{i-1}| + O(1)$. Zatem łącznie wykonamy liniowo wiele operacji. □

3.3 Algorytm Ukkonena

Motywacja - ulepszenie algorytmu naiwnego. Budujemy nieskompresowane drzewo sufiksowe dla słowa $T = t_0t_1t_2...t_{n-1}$. W k-tej fazie dodajemy do drzewa T_{k-1} krawędzie z etykietą t_k, tak aby otrzymać drzewo T_k. Trzymamy listę miejsc gdzie kończą się sufiksy i przedłużamy je o t_k.

Algorytm korzysta z 3 własności:

- Za każdym razem gdy przedłużamy sufiks dodając nowy liść do drzewa sufiksowego, możemy o tym sufiksie już zapomnieć. Przy założeniu opisu krawędzi skompresowanych za pomocą dwóch liczb (początek i koniec jakiegoś podsłowa w T) możemy jako drugiej liczby użyć $+\infty$. Zachowamy własność online algorytmu. Taka krawędź reprezentować będzie dowolny tekst zaczynający się w określonym punkcie.

- Jeśli rozważając kolejne sufiksy natrafimy na taki, którego nie musimy przedłużać, bo odpowiednia krawędź znajduje się już w drzewie, to nie musimy też przeglądać dalszych, ponieważ dla nich także odpowiednia krawędź występuje.

- Aby w razie potrzeby namierzyć kolejne miejsce wymagające rzeczywistej aktualizacji, tzn. dodania krawędzi-liścia, można zastosować w tym celu ten sam trik, co w algorytmie McCreighta. Zapamiętujemy jedynie dowiązania sufiksowe wierzchołków jawnych, a jeśli chcemy znaleźć dowiązanie wierzchołka niejawnego, to najpierw przechodzimy po dowiązaniu sufiksowym jego (jawnego) rodzica w drzewie, a następnie wykonujemy operację $fastFind$.

3.3.1 Algorytm

Pseudokod algorytmu został zaprezentowany jako algorytm 2.

Algorithm 2 Algorytm Ukkonena - konstrukcja online

$T \leftarrow root$
$L \leftarrow root$
for $i = 1 \rightarrow n$ **do**
  for $j = 1 \rightarrow n$ **do**
    $v \leftarrow L[j]$
    if nie istnieje krawędź w drzewie T o etykiecie x_i wychodząca z v **then**
      dodaj taką krawędź
      $L[j] \leftarrow v'$, gdzie v' jest wierzchołkiem docelowym krawędzi wychodzącej z v etykietowanej x_i;
    end if
  end for
  $L.push(root)$
end for

3.3.2 Złożoność

Twierdzenie 2. *Algorytm Ukkonena buduje drzewo sufiksowe tekstu o długości n w czasie $O(n)$, przy założeniu $|\Sigma| = O(1)$.*

Dowód. Obrotów wewnętrznej pętli while mamy liniowo wiele - każdy obieg odpowiada dodaniu jednego liścia, a tych jest $O(n)$. Wszystkie operacje w pętli poza $fastFind$ wykonujemy w czasie stałym. Należy zatem rozważyć łączną pracę wykonaną przez wszystkie wywołania $fastFind$. □

4 Podsumowanie

4.1 Zastosowania

Drzewa sufiksowe znajdują zastosowanie między innymi w bioinformatyce, gdzie służą do analizy łańcuchów DNA i sekwencji aminokwasów w białkach, oraz w kompresji danych (modyfikacje kompresji LZW).

Literatura

[1] Prof. Erik Demaine, http://courses.csail.mit.edu/6.851/spring12/scribe/lec16.pdf

[2] http://wazniak.mimuw.edu.pl/index.php?title=Algorytmy_i_struktury_danych