Permuted Longest-Common-Prefix Array

Demian Banakh

1 Notacja

- $a * b$ znaczykatenację słów a i b.
- Tekst wejściowy oznaczam $t[1..n]$
- Sufiks i znaczy $t[i..n]$
- Tablica sufiksowa dla t to SA$[1..n]$

$$t[\text{SA}[1]..n] < t[\text{SA}[2]..n] < \cdots < t[\text{SA}[n]..n]$$

- Funkcja $lcp(x, y)$ zwraca najkrótszy wspólny prefiks słów x i y. Dla wygody będę oznaczał

$$lcp(i, j) := lcp(t[i..n], t[j..n])$$

- Tablica LCP$[1..n]$ zdefiniowana przez t i SA to

$$\text{LCP}[j] = lcp(\text{SA}[j-1], \text{SA}[j])$$

- Tablica PLCP$[1..n]$ jest permutacją tablicy LCP zdefiniowaną wzorem

$$\text{PLCP}[\text{SA}[i]] = \text{LCP}[i]$$

2 Algorytmy i dowody

Kluczowa własność PLCP:

Lemma 2.1. $\text{PLCP}[i] \geqslant \text{PLCP}[i-1] - 1$ dla każdego $i > 1$.

Dowód. Niech j, j' takie że $\mathrm{SA}[j] = i$ oraz $\mathrm{SA}[j'] = i - 1$. Mamy z definicji

$$\mathrm{PLCP}[i] = \mathrm{LCP}[j] = lcp(\mathrm{SA}[j-1], i)$$
$$\mathrm{PLCP}[i-1] = \mathrm{LCP}[j'] = lcp(\mathrm{SA}[j'-1], i-1)$$

Rozważmy następujące sufiksy

$$A = t[\mathrm{SA}[j-1]-1..n] = t[\mathrm{SA}[j-1]-1] * t[\mathrm{SA}[j-1]..n]$$
$$B = t[\mathrm{SA}[j'-1]..n] = t[\mathrm{SA}[j'-1]] * t[\mathrm{SA}[j'-1]+1..n]$$
$$C = t[i-1..n] = t[i-1] * t[i..n]$$
$$B \leqslant C \text{ (poprzedni leksykograficznie)}$$

Oczywiście $lcp(\mathrm{SA}[j'-1]+1, i) \geqslant lcp(\mathrm{SA}[j'-1], i-1) - 1$, przy czym równość jest wtw gdy $t[\mathrm{SA}[j'-1]] = t[i-1]$. Skoro nie ma sufiksów pomiędzy $\mathrm{SA}[j-1]$ a $\mathrm{SA}[j] = i$ w porządku leksykograficznym, to

$$t[\mathrm{SA}[j'-1]+1..n] \leqslant t[\mathrm{SA}[j-1]..n] < t[i..n]$$

Zatem

$$lcp(\mathrm{SA}[j-1], i) \geqslant lcp(\mathrm{SA}[j'-1]+1, i) \geqslant lcp(\mathrm{SA}[j'-1], i-1) - 1$$

□

Ten lemat prawie wprost prowadzi do wydajnego Algorytmu A obliczenia PLCP: mając obliczoną wartość $\mathrm{PLCP}[i-1]$, odejmujemy 1 i porównujemy sufiksy znak po znaku dopóki są równe. Oczywiście musimy wiedzieć który to jest sufiks $\mathrm{SA}[j-1]$; do tego celu liczymy dodatkową tablicę $\Phi[\mathrm{SA}[j]] = \mathrm{SA}[j-1]$.

Algorithm 1 Algorytm A

Require: SA and text
Ensure: PLCP

```
for i = 1 to n do
  Φ[SA[i]] = SA[i − 1]
end for
l ← 0
for i = 1 to n do
  s ← Φ[i]
  while text[i + l] = text[s + l] do
    l ← l + 1
  end while
  PLCP[i] ← l
  l ← max(l − 1, 0)
end for
```

Łatwo widać, że złożoność czasowa Algortymu A jest $O(n)$, bo zmniejszamy l o 1 co najwyżej n razy, a maksymalna wartość l jest co najwyżej n. Z punktu widzenia pamięci, potrzebuje dodatkowej tablicy Φ rozmiaru n.

W dość naturalny sposób można ulepszyć kontrolę nad space-time tradeoff modyfikując algorytm tak, żeby produkował tylko co q-ty element PLCP (wystarczy obliczyć tylko co q-ty element Φ; naiwne liczenie PLCP$[i]$ zaczynamy od (PLCP$[i-1]-q$) - wynika z Lematu 1), a obliczenie dowolnego elementu PLCP na bazie co q-tego było w czasie amortyzowanym $O(q)$ (dla dowolnego i szukamy najbliższych takich k, $k+q$ na których wartość PLCP jest znana; wtedy obliczamy naiwnie PLCP$[i]$ w interwale zadanym przez PLCP$[k]$ i PLCP$[k+q]$, którego długość jest średnio $O(q)$).

Definition 2.1. Nazywamy wartość $\mathrm{PLCP}[i] = lcp(i, \Phi[i])$ redukowalną, gdy $t[i-1] = t[\Phi[i]-1]$.

Lemma 2.2. *Jeśli $PLCP[i]$ jest redukowalna, to $PLCP[i] = PLCP[i-1]-1$.*

Dowód. Pokażemy to, kontynuując dowód Lematu 1 z dodatkowym założeniem

$$t[i-1] = t[\mathrm{SA}[j-1]-1]$$

W takim razie mamy $A < C$, ponieważ $\mathrm{SA}[j-1]$ i $\mathrm{SA}[j] = i$ to są kolejne sufiksy. Skoro B i C to też kolejne sufiksy

$$A \leqslant B < C$$

Skoro pierwsze litery A i C są równe, pierwsza litera B musi być taka sama. Oznaczmy A', B', C' odpowiednio A, B, C bez pierwszych liter. Mamy

$$A' \leqslant B' < C'$$

Ale A' i C' to kolejne sufiksy, więc $A' = B'$, oraz $\mathrm{SA}[j-1] = \mathrm{SA}[j'-1]+1$. W końcu

$$lcp(\mathrm{SA}[j-1], i) = lcp(\mathrm{SA}[j'-1]+1, i) = lcp(\mathrm{SA}[j'-1], i-1) - 1$$

ponieważ $t[i-1] = t[\mathrm{SA}[j-1]-1] = t[\mathrm{SA}[j'-1]]$. □

Wnioskujemy z tego lematu, że nieredukowalne wartości PLCP$[i]$ jednoznacznie wyznaczają pozostałe, zatem algorytm B obliczenia PLCP najpierw wylicza wszystkie nieredukowalne wartości, potem dopełnia pozostałe.

Algorithm 2 Algorytm B

Require: SA and text
Ensure: PLCP
  for $i = 1$ **to** $n - 1$ **do**
    $j \leftarrow \mathrm{SA}[i]$
    $k \leftarrow \mathrm{SA}[i+1]$
    if $text[j-1] = text[k-1]$ **then**
      $\mathrm{PLCP}[k] \leftarrow lcp(j,k)$
    end if
  end for
  for $i = 1$ **to** $n - 1$ **do**
    if $\mathrm{PLCP}[i+1] < \mathrm{PLCP}[i] - 1$ **then**
      $\mathrm{PLCP}[i+1] \leftarrow \mathrm{PLCP}[i] - 1$
    end if
  end for

Lemma 2.3. Suma wszystkich nieredukowalnych wartości lcp jest $\leqslant 2n \log n$.

Dowód. Niech $l = \mathrm{PLCP}[i] = lcp(i,j)$, gdzie $j = \Phi[i]$, jest nieredukowalną wartością. Zatem

$$t[i-1] \neq t[j-1]$$
$$t[i..i+l-1] = t[j..j+l-1]$$
$$t[i+l] \neq t[j+l]$$

Dla każdego $0 \leqslant k \leqslant l-1$, będziemy przydzielać koszt 1 do pary $t[i+k] = t[j+k]$ w następujący sposób.

Rozpatrzmy drzewo sufiksowe słowa $\bar{t}$; niech v_{i+k} i v_{j+k} będą liśćmi odpowiadającymi prefiksom $t[1..i+k]$ i $t[1..j+k]$. Najniższy wspólny przodek u węzłów v_{i+k} i v_{j+k} odpowiada $t[i..i+k]$ - te prefiksy zgadzają się na dokładnie $(k+1)$ znaków od końca. Jeśli v_{i+k} jest w mniejszym poddrzewie u, to koszt pary $t[i+k] = t[j+k]$ przydzielamy do węzła v_{i+k}, wpp do v_{j+k}. Gdy v_{i+k} został wybrany, nazwiemy u *drogim przodkiem* węzła v_{i+k}, a v_{j+k} *drogim bratem* węzła v_{i+k} w odniesieniu do u, wpp analogicznie.

Teraz wystarczy pokazać, że każdy liść ma koszt co najwyżej $2 \log n$. W szczególności, pokażmy, że (a) każdy liść ma co najwyżej $\log n$ drogich przodków, i (b) w odniesieniu do każdego z tych drogich przodków dany liść ma co najwyżej 2 drogich braci.

- Rozważmy ścieżkę od v do korzenia. Z konstrukcji, przy każdym drogim przodku węzła v na ścieżce, poddrzewo rośnie przynajmniej w 2 razy. Zatem może być nie więcej niż $\log n$ takich przodków.

- Niech u to drogi przodek węzła v i w to drogi brat węzła v w odniesieniu do u. Mamy, że v, u, w odpowiadają

  $$t[1..i+k], t[i..i+k], t[1..j+k] \text{ dla pewnych } i, j \text{ tż } i = \Phi[j] \text{ lub } j = \Phi[i]$$

  Bez ograniczenia ogólności, niech $i = \Phi[j]$. Załóżmy, że istnieje inny drogi brat $w' \neq w$ węzła v w odniesieniu do u. Teraz w' musi odpowiadać $t[1..j'+k]$ dla $j' = \Phi[i]$ (ponieważ $i = \Phi[j] \neq \Phi[j']$). Od razu widać, że trzeciego takiego brata nie może istnieć.

Suma kosztów wszystkich liści w drzewie (równa sumie wszystkich nieredukowalnych wartości lcp) jest ograniczona przez $2n \log n$. □

Z tego lematu wnioskujemy, że złożoność czasowa Algorytmu B jest $O(n \log n)$, natomiast z punktu widzenia pamięci - używa $O(1)$ poza samą tablicą PLCP.

Podobnie jak wcześniej, ten algorytm da się w naturalny sposób zmodyfikować tak, żeby produkował co q-ty element PLCP, a obliczenie dowolnego na bazie co q-tego zajmowało średnio $O(q)$.

Warto dodać, że przedstawione algorytmy obliczenia PLCP są znacznie szybsze w praktyce, niż standardowe algorytmy obliczenia LCP. Zaletą Algorytmu A jest bardzo skuteczne wykorzystanie własności *locality of reference*, drugiego - wymagania pamięciowe (jeśli przedstawić PLCP jako bit-tablicę, potencjalnie można zmieścić się 3n bitów pamięci).