Algorytm Faracha

Bartosz Wodziński

May 27, 2020

Cel algorytmu i format wejścia

Wejściem do algorytmu jest słowo w o długości n nad alfabetem $\Sigma = [n]$ - liczb naturalnych od 1 do n. Zakładać będziemy, że $n = 2^k$ - jeśli nie, to można słowo w wydłużyć o czynnik liniowy, tak żeby ta równość zachodziła, dodając na koniec odpowiednią ilość $'\$'$, a po wykonaniu algorytmu "uciąć" z drzewa sufiksowego poddrzewa odpowiadające tym symbolom.

Algorytm tworzy drzewo sufiksowe dla tego słowa w czasie $O(n)$. Od drzewa wymagane jest, aby dla każdego wierzchołka, krawędzie prowadzące do jego dzieci posortowane były leksykograficznie względem ich etykiet (w przypadku liczb naturalnych porządek leksykograficzny rozumiemy jako zdefiniowany przez zwykłą relację $<$).

Idea algorytmu

Algorytm Faracha działa rekurencyjnie i każdy jego etap można zapisać w trzech krokach:

1. Zbuduj drzewo sufiksowe T_o dla *nieparzystych* sufiksów, czyli takich, które zaczynają się od nieparzystego indeksu (numerujemy od 1).
2. Korzystając z drzewa T_o, zbuduj drzewo T_e - drzewo sufiksowe dla *parzystych* sufiksów (bez wywołania rekurencyjnego!).
3. Połącz drzewa T_o i T_e w jedno drzewo T.

Jeśli kroki 2. i 3 będziemy w stanie wykonać w liniowej złożoności, to wówczas cały algorytm będzie miał złożoność:

$$T(n) = T\left(\frac{n}{2}\right) + O(n),$$

a więc liniową.
Aby uniknąć dodatkowego wywołania rekurencyjnego w równaniu (co zwiększyłoby złożoność do $O(nlgn)$), drzewo T_e konstruowane będzie bezpośrednio z drzewa T_o.

Tworzenie drzewa T_o

Tworzenie drzewa nieparzystych sufiksów przebiega w kilku krokach:

1. $\forall i \in [\frac{n}{2}]$ stwórz parę $\langle w[2i-1], w[2i]\rangle$ i wszystkie takie pary umieść w liście S' i każdą z nich zamień na jej *rangę* (indeks w posortowanej leksykograficznie liście). Na przykład, dla słowa `121112212221`, lista S' wygląda tak:
`[(1,2), (1,1), (1,2), (2,1), (2,2), (2,1), #]`, a po zamianie elementów na ich rangi, tak:
`[2,1,2,3,4,3,#]`.
2. Rekurencyjnie oblicz $T_{S'}$ - drzewo sufiksowe dla słowa S' oraz jego tablicę sufiksową składającą się z $A_{T_{S'}}$ - posortowanej tablicy sufiksów i z $LCP_{T_{S'}}$ - tablicy najdłuższych wspólnych prefiksów dla $A_{T_{S'}}$.

3. Korzystając z $A_{T_{S'}}$, oblicz A_{T_o} w następujący sposób:

$$A_{T_o}[i] = 2A_{T_{S'}}[i] - 1,$$

co bazuje na obserwacji, że każdy nieparzysty sufiks $w[2i-1]...w[n]$ jest równoważny sufiksowi $S'[i]...S'[\frac{n}{2}]$ (co wynika z konstrukcji S'), a zatem leksykograficzny porządek $A_{T_{S'}}$ jest taki sam jak porządek A_{T_o}. Jedyne co trzeba zmienić, to indeksy, odwracając przekształcenie z punktu 1.: $\langle w[2i-1], w[2i]\rangle \to S'[i]$.

4. Oblicz $LCP_{A_{T_o}}$, korzystając z $LCP_{A_{T_{S'}}}$, według wzoru:

$$LCP_{T_o}[i] = 2LCP_{T_{S'}}[i] + \begin{cases} 1 \text{ if } w[A_{T_o}[i] + 2LCP_{T_{S'}}] = w[A_{T_o}[i+1] + 2LCP_{T_{S'}}[i]] \\ 0 \text{ otherwise} \end{cases},$$

co również wynika z konstrukcji słowa S'.

5. Na podstawie A_{T_O} i LCP_{T_o} skonstruuj drzewo T_o.

Tworzenie drzewa T_e

1. Wstępnie przetwórz drzewo T_o (w czasie liniowym od jego rozmiaru) tak aby dało się odpowiadać na zapytania o `lca` na tym drzewie w czasie stałym.

2. Korzystając z obserwacji, że każdy parzysty sufiks to nieparzysty sufiks poprzedzony jednym znakiem, stwórz A_{T_e}. W tym kroku możemy wykorzystać obliczoną już listę A_{T_o}, "dokleić" odpowiedni znak przed każdym elementem A_{T_o} i użyć sortowania pozycyjnego tylko dla tego pierwszego znaku.

3. Oblicz tablicę LCP_{T_e} korzystając z zapytań o `lcp(word(a),word(b))`, czyli zapytań o `lca(a,b)` w drzewie sufiksowym:

$$lcp(w[2i,n], w[2j,n]) = \begin{cases} lcp(w[2i+1,n], w[2j+1,n]) + 1 \text{ if } w[2i] = w[2j] \\ 0 \text{ otherwise} \end{cases},$$

4. Stwórz T_e w oparciu o A_{T_e} i LCP_{T_e}.

Łączenie drzew T_o i T_e

Zauważmy, że aby stworzyć drzewo T z drzew T_o i T_e, wystarczy zbudować tablice A_T i LCP_T. Do uzyskania tych tablic potrzebujemy móc szybko (w czasie stałym) odpowiadać na pytania o najdłuższy wspólny prefiks dwóch sąsiadujących sufiksów w tablicy A_T. Umożliwiającą nam to wyrocznię uzyskamy poprzez *zachłanne* połączenie drzew T_o i T_e.

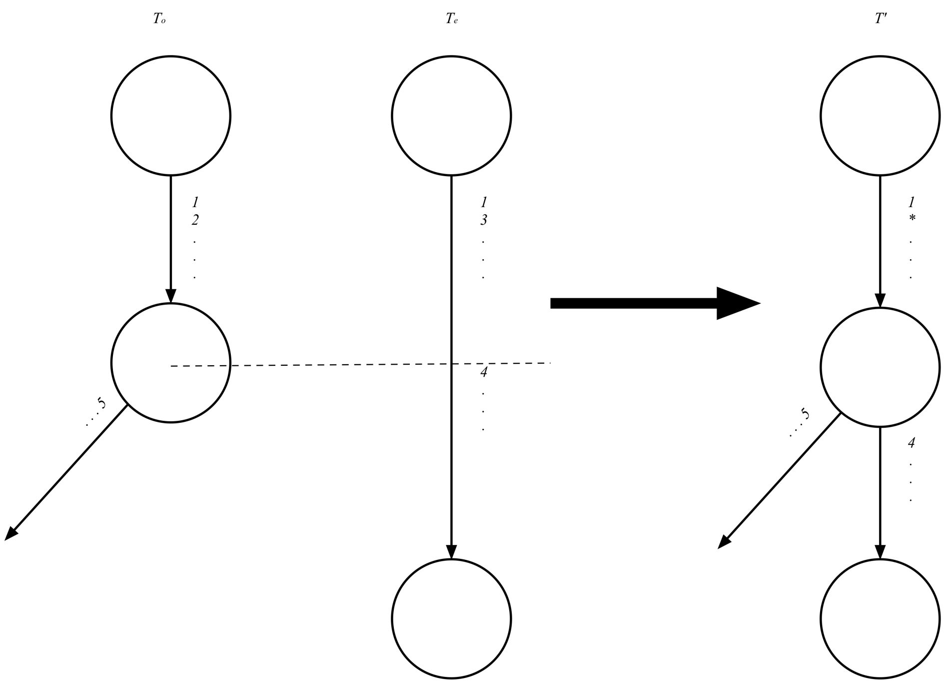
Zachłanne połączenie T' rozumiemy jako takie połączenie dwóch drzew (T_e i T_o), w którym:

- Każdy wierzchołek z tych drzew ma odpowiadający sobie wierzchołek w drzewie T', a ponadto, jeśli $e \in T_e$, $o \in T_o$ i `word(e) = word(o)`, to wierzchołkom e i o odpowiada ten sam wierzchołek $t \in T$.
  Wierzchołek u *odpowiada* wierzchołkowi $v \in T_o \vee v \in T_e$ wtedy i tylko wtedy gdy $|word(u)| = |word(v)|$, a ponadto dla każdego wierzchołka a z T_o (lub T_e), który jest na drodze z korzenia do wierzchołka v zachodzi: $word(v)[|word(a)| + 1] = word(u)[|word(a)| + 1]$, czyli $word(v)$ i $word(u)$ są tej samej długości i zgadzają się przynajmniej na indeksach, które odpowiadają pierwszym znakom na każdej krawędzi na drodze z korzenia do v.

- Z każdego wierzchołka $t \in T$, dla dowolnych dwóch etykiet w, v krawędzi z niego wychodzących zachodzi: $w[0] \neq v[0]$ ($w[1] \neq v[1]$ przy numeracji od 1).

- Drzewo T' nie zawiera żadnych innych wierzchołków.

W skrócie można powiedzieć, że T' to drzewo, które powstaje dokładnie w ten sposób co naiwnie wykonane poprawne połączenie T_o i T_e do T, z tą tylko różnicą, że zamiast porónywać wszystkie znaki na odpowiednich krawędziach (powiedzmy u i v) (i ewentualnie dzielić je jeśli różnią się na jakiejś pozycji), porównujemy tylko pierwsze znaki, a podział robimy tylko w miejscu, w którym skończy się krótsza z nich. Procedurę tworzenia T' (czyli "nałożenia na siebie" drzew T_o i T_e patrząc tylko na pierwszy znak krawędzi) przedstawia poniższy rysunek:



Wiedząc jak jest konstruowane drzewo T', zapiszmy kroki do stworzenia poprawnego drzewa T:

1. Stwórz T' tak jak zostało to wyżej opisane.

2. Wierzchołek drzewa T', który ma odpowiadający wierzchołek w T_o nazwijmy *nieparzystym*, a jeśli ma odpowiadający wierzchołek w T_e, to nazwijmy go *parzystym*. Pewne wierzchołki (np. korzeń) mogą być zarówno parzyste jak i nieparzyste.
   Teraz, dla każdego wierzchołka $u \in T'$ znajdź parę wierzchołków (jeśli istnieje) (a, b), które są jego potomkami i liśćmi drzewa, a ponadto a jest parzysty, b jest nieparzysty oraz `lca(a,b) = u`. Załóżmy, że `word(a) = w[2i,n]`, `word(b) = w[2j-1,n]`. Dodatkowo, każdemu liściowi c takiemu, że `word(c) = w[k,n]` nadajmy etykietę l_k. Zgodnie z tą konwencją $a = l_{2i}$, $b = l_{2j-1}$.
   Zdefiniujmy funkcję d taką, że $d(u) = lca(l_{2i+1}, l_{2j})$ i policzmy ją dla każdego u. Zauważmy, że jeśli T' byłoby poprawnym drzewem, to $d(u) = suffix\_link(u)$.

3. Zakładając, że funkcja d definiuje drzewo, dla każdego wierzchołka u policz jego głębokość $L(u)$ w tym drzewie. Skorzystaj z faktu, że $lcp(w[2i, n], w[2j-1]) = L(lca(l_{2i}, l_{2j-1}))$ i stwórz tablicę LCP_T.

Dowód poprawności

Wystarczy udowodnić poprawność punktów 2. i 3.

Twierdzenie. *Najdłuższy wspólny prefiks dowolnej pary liści (a, b), takich, że a - parzysty, b - nieparzysty, i takich, że $\mathtt{lca}(a, b) = u$, jest taki sam.*

Dowód. Weźmy wierzchołek parzysty u mający zarówno parzystego jak i nieparzystego potomka. Weźmy teraz dwóch parzystych potomków (być może takich samych) u: l_{2i}, l_{2j}. Wówczas, $\mathtt{lcp}(w[2i, n], w[2j, n]) \geq |word(u)|$, ponieważ u, l_{2i}, l_{2j} były w drzewie T_e.

Teraz, weźmy parę $l_{2i'-1}$, $l_{2j'-1}$ potomków u. Zachodzi wtedy $\mathtt{lcp}(w[2i' - 1, n], w[2j' - 1, n]) \geq |word(u)|$. Jest tak, ponieważ, jeśli u był również w T_o, to działa ten sam argument co poprzednio. Jeśli nie był w T_o, to ponieważ $l_{2i'-1}$, $l_{2j'-1}$ są liśćmi, to musi istnieć wierzchołek $u' = \mathtt{lca}(l_{2i'-1}, l_{2j'-1}) \in T_o$, który jest potomkiem u w T'.

Rozważmy teraz parę liści $l_{2i''}$ i $l_{2j''-1}$ w T', takich że $u = \mathtt{lca}(l_{2i''}, l_{2j''-1})$. Zachodzi wtedy $\mathtt{lcp}(w[2i'', n], w[2j'' - 1, n]) \leq |word(u)|$. Wynika to z faktu, że jeśli byłoby inaczej, to wtedy więcej niż jedna krawędź wychodząca z u zaczynała by się tym samym znakiem, co jednak w T' nie ma miejsca. Analogiczne rozumowanie możemy przeprowaddzić dla u będącego wierzchołkiem nieparzystym i wówczas każda z powyższych nierówności również będzie zachodziła.

Weźmy w końcu liście $l_{2i'}$,, $l_{2i''}$, $l_{2j'-1}$, $l_{2j''-1}$ takie, że $\mathtt{lca}(l_{2i'}, l_{2j'-1}) = \mathtt{lca}(l_{2i''}, l_{2j''-1}) = u$. Wtedy, $\mathtt{lcp}(w[2i', n], w[2j' - 1, n]) = k \leq |word(u)|$, co udowodniliśmy przed chwilą. Wiemy jednak, że $\mathtt{lcp}(w[2i', n], w[2i'', n]) \geq |word(u)| \geq k$, a zatem musi być $\mathtt{lcp}(w[2i'', n], w[2j' - 1, n]) = k$. Podobnie, $\mathtt{lcp}(w[2i'', n], w[2j'' - 1, n]) = k$. Wobec tego:
$\mathtt{lcp}(w[2i', n], w[2j' - 1, n]) = \mathtt{lcp}(w[2i'', n], w[2j'' - 1, n]) = k$. $\square$

Twierdzenie. *Funkcja d definiuje drzewo na wierzchołkach T', a ponadto dla dowolnych l_{2i} i l_{2j-1} zachodzi $L(\mathtt{lca}(l_{2i}, l_{2j-1})) = \mathtt{lcp}(w[2i, n], w[2j - 1, n])$.*

Dowód. Dowód jest indukcyjny ze względu na długość `lcp`. Jeśli `lcp`$word[2i,n], word[2j-1,n]) = 0$, to wówczas $word[2i,n]$ i $word[2j-1,n]$ różnią się na pierwszym znaku, więc $\texttt{lca}(l_{2i}, l_{2j-1}) =$ `root`, co wynika ze sposobu konstruowania drzewa T'.

Załóżmy teraz, że twierdzenie zachodzi dla pary sufiksów parzystych i nieparzystych, takich, że ich $\texttt{lcp} < k$. Niech l_{2i}, l_{2j-1} będą liśćmi takimi, że $\texttt{lca}(w[2i,n], w[2j-1,n]) = k > 0$. Ponadto, niech $u = \texttt{lca}(l_{2i}, l_{2j-1})$. Wtedy u nie jest korzeniem, bo $k > 0$.

Niech teraz $l_{2i'}$, $l_{2j'-1}$ będą liśćmi użytymi do zdefiniowania $d(u)$. Zatem, z definicji, $d(u) = \texttt{lca}(l_{2i'+1}, l_{2j'})$.

Z założeń indukcyjnych wiemy, że funkcja d definiuje drzewo na dotychczas rozważonych wierzchołkach, a ponadto, że funkcja L jest zdefiniowana na tych wierzchołkach i że zwraca poprawne wartości `lcp`. Zatem, z faktu, że funkcja d definiuje drzewo, wiemy że $L(u) = 1 + L(d(u))$. Ponadto, z faktu, że funkcja L zwraca poprawne wartości dla dotychczas rozważonych wierzchołków wiemy, że $1 + L(d(u)) = 1 + \texttt{lcp}(w[2i'+1,n], w[2j',n])$. Teraz, ponieważ $k > 0$ (w szczególności oznacza to, że $w[2i'] = w[2j'-1]$), zachodzi $1 + \texttt{lcp}(w[2i'+1,n], w[2j',n]) = \texttt{lcp}(w[2i',n], w[2j'-1,n])$. A z poprzedniego twierdzenia wiemy, że $\texttt{lcp}(w[2i',n], w[2j'-1,n]) = \texttt{lcp}(w[2i,n], w[2j-1,n])$.

Wobec tego, $L(u) = \texttt{lcp}(w[2i,n], w[2j-1,n])$, a ponadto struktura zdefiniowana przez funkcję d jest drzewem. □

Złożoność

Tworzenie T_o

1. Sortowanie pozycyjne dla par, w których maksymalna liczba jest mniejsza lub równa n (tak jest, bo początkowe słowo spełnia to założenie, a każde następne powstaje przez zastąpienie elementu przez jego rangę) zajmuje $O(n)$ czasu.

2. Jeśli wszystkie inne kroki będą liniowe, to ten krok zajmie $T(n) = T(\frac{n}{2}) + O(n) = O(n)$ czasu.

3. Każdy element tablicy A_{T_o} liczymy w czasie stałym, więc ten krok zajmuje $O(n)$ czasu.

4. Podobnie jak w poprzednim punkcie, mamy stały czas na przetworzenie jednego elementu tablicy, więc całość robimy w $O(n)$.

5. Jest możliwe skonstruowanie drzewa sufiksowego w czasie liniowym z tablic A i LCP.
   Idea: wstawiamy kolejne sufiksy w kolejności leksykograficznej. Po wstawieniu każdego sufiksu, zatrzymujemy się w liściu, który go reprezentuje i dopóki $\texttt{lcp}(word(v), next\_suffix) < |word(v)|$ wykonujemy, $v = v.parent$, a następnie dodajemy nową krawędź z wierzchołka v albo dzielimy już wychodzącą.

Tworzenie T_e

1. Da się w czasie liniowym zrobić preprocessing drzewa tak aby w czasie stałym odpowiadać na zapytania `lca`. Jest to nietrywialny algorytm,

który jest opisany w `https://www.ics.uci.edu/~eppstein/261/BenFar-LCA-00.pdf`.

2. Sortowanie pozycyjne na liczbach nieprzekraczających n działą w $O(n)$.

3. Ponieważ zapytania o `lcp` przetwarzane są w czasie stałym, czas działania tego kroku to $O(n)$.

4. Analogicznie jak dla T_o - $O(n)$.

Łączenie drzew T_o i T_e

1. Ten krok to zwykły DFS działający liniowo względem sumarycznej ilości wierzchołków w drzewach T_o i T_e, a ponieważ są to poprawne drzewa sufiksowe, to mają liniowe rozmiary względem $O(\frac{n}{2})$.

2. Odpowiednią parę liści dla każdego wierzchołka z T' możemy znaleźć przy pomocy jednego DFS-a, korzystając z obliczonych wcześniej wartości dla potomków tego wierzchołka.

   Funkcję d liczymy dla każdego wierzchołka w czasie stałym, po wcześniejszym liniowym przetworzeniu drzewa T' do zapytań o `lca`.

3. Głębokość w drzewie liczymy ponownie używając algorytmu DFS i korzystając z obliczonych wartości, każdy element LCP_T liczymy w czasie stałym. Odtworzenie drzewa z tablic A_T i LCP_T wykonujemy w czasie $O(n)$, jak wyżej.